universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Object Tracking in the Cloud“

verfasst von / submitted by

Matthias Klan, B.Sc.

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Diplom-Ingenieur (Dipl. Ing.)

Wien, 2017 / Vienna 2017

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 935
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Medieninformatik UG2002
Betreut von / Supervisor:	Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas
Mitbetreut von / Co-Supervisor:	Mag. Dr. Maia Rohm

Erklärung zur Verfassung der Arbeit

Matthias Klan
Untere Weißgerberstraße 10/19, 1030 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 13. Dezember 2017

Matthias Klan

Acknowledgements

First of all I would like to thank Univ.-Prof. Dipl.-Ing. Dr. Wolfgang Klas for offering me to write this thesis under his supervision. I would also like to thank my secondary adviser Mag. Dr. Maia Rohm for her constant supervision over the whole period of the thesis' creation process. Her support in terms of constructive expert feedback and continuous revision was an immense help and inspiration for me. I would like to especially thank my parents Jana and Vitek for supporting me unconditionally my entire life and thus enabling me to achieve the station of life I am currently at. Finally I would like to thank my girlfriend Neele for her steady encouragement and regard.

Abstract

Recently, software applications are moving from the classic desktop to the web. A crucial advantage of this process is the possibility to outsource heavy computation tasks to decoupled hardware in the cloud. Such tasks can be commonly found in the computer vision domain, which tries to imitate the human vision system in order to recognize specific features in visual media. A specific computer vision task is the object tracking, where algorithms aim to locate an object of interest in consecutive frames of a video. This functionality can be used, for example, for traffic monitoring or surveillance. There are many algorithms addressing the problem of object tracking. They are mostly loosely published or integrated in libraries such as OpenCV. The coding of an extra Infrastructure to bring these algorithms into the cloud must be done by the developer himself. Existing solutions such as 'Wirewax' are not flexible enough and restrict the usage to their ecosystem. The aim of this thesis is to design and develop a web service to support developers in building cloud-based object tracking applications more efficiently. In the process of design, three different object tracking algorithms are analyzed and evaluated (Tracking-Learning-Detection, Consensus-based matching and tracking of key-points for object tracking and correlation tracker). Additionally, a suitable set of communication protocols is considered. The resulting concept is a division of the single object tracking algorithms into independent micro services, which are communicating via the Advanced Message Queuing Protocol with a web server. This web server in turn communicates using WebSockets with the clients of the web service in order to orchestrate the requests and the results between the client and the different algorithms. The outcome is a web service that can be easily spun up on a web server due to containerization of the software. Furthermore, the system is able

to load balance requests between multiple instances of a given algorithm. New algorithms can be added later on and a JavaScript library enables a hassle-free integration into browser-based web applications.

Kurzfassung

In letzter Zeit verlagern sich Softwareanwendungen vom klassischen Desktop ins Web. Ein wesentlicher Vorteil von diesem Prozess ist die Möglichkeit berechnungsintensive Aufgaben an entkoppelte Hardware in der Cloud auszulagern. Solche Aufgaben können üblicherweise im Bereich von Computervision gefunden werden, welcher versucht das menschliche Sehsystem zu immitieren, um bestimmte Eigenschaften in visuellen Medien zu erkennen. Objecttracking ist ein Teilgebiet von Computervision, wo Algorithmen versuchen definierte Objekte in aufeinanderfolgenden Einzelbildern von Videos zu lokalisieren. Diese Funktionalität kann zum Beispiel in der Verkehrsbeobachtung oder der öffentlichen Überwachung benutzt werden. Es gibt zahlreiche Algorithmen, die dieses Problem versuchen zu lösen. Diese sind meistens vereinzelt veröffentlicht oder in Bibliotheken wie OpenCV integriert. Das Entwickeln von einer zusätzlichen Infrastruktur um diese Algorithmen in die Cloud zu bringen muss jedoch von dem Entwickler selbst geleistet werden. Bestehende Lösungen wie 'Wirefax' sind nicht flexibel genug und beschränken die Nutzung auf deren Ökosystem. Das Ziel dieser Arbeit ist es einen Webservice zu konzipieren und zu entwickeln, welcher Entwicklern beim Erstellen von cloudbasierten Objecttracking-Applikationen unterstützen soll. Im Zuge der Konzeption werden drei unterschiedliche Objecttracking-Algorithmen analysiert und evaluiert (Tracking-Learning-Detection, Consensus-based matching and tracking of key-points for object tracking and correlation tracker). Des Weiteren wird eine Zusammenstellung von Kommunikationsprotokollen in Betracht gezogen. Das resultierende Konzept ist eine Zerteilung der einzelnen Objecttracking-Algorithmen in unabhängige Microservices, die über das Advanced Message Queuing Protocol mit dem Webserver kommunizieren. Dieser Webserver kommuniziert wiederrum

mithilfe von WebSockets mit den Clients vom Webservice, um die Anfragen und Ergebnisse zwischen diesen und den jeweiligen Algorithmen zu orchestrieren. Das Ergebnis ist ein Websevice, den man mithilfe von Containerisierung einfach auf einem beliebigen Webserver starten kann. Des Weiteren lassen sich Anfragen auf mehrere Instanzen von den besagten Algorithmen verteilen. Neue Algorithmen können im Nachhinein hinzugefügt werden und eine JavaScript Bibliothek erlaubt die mühelose Integration in browserbasierende Webapplikationen.

Contents

CHAPTER 1

Introduction

"When wireless is perfectly applied the whole earth will be converted into a huge brain, which in fact it is, all things being particles of a real and rhythmic whole. We shall be able to communicate with one another instantly, irrespective of distance." - Nicola Tesla (1926) [16]

The visionary futurist and inventor Nicola Tesla was proven right. Thanks to his invention of the polyphase alternating current, our society resides in a digital era which most of us nowadays take for granted. Everyone is permanently connected trough the internet with the rest of the world. Moreover, the entire knowledge of the humankind is potentially accessible trough a small device in our pocket. The information is stored in central repositories (also known as the cloud) and transmitted over electricity on demand.

In this thesis we go a step further and use external machines in the cloud not only to hold static data. We rather use their computation abilities to retrieve context-specific data depending on the input we provide. This is a fundamental capability of every computer and in this work we outsource the work from our device to an external, possibly more powerful computer. This concept has several advantages, such as monetarization of services or the delegation of concerns from the user to the provider. In general we outsource time and computation intensive tasks in order to unload the user's device.

Those intensive tasks can be found, for example, in the scientific field of computer vision. Computer vision algorithms try to mimic the human visual system and can be used, e.g., to recognize specific objects in an unknown scene. Recognition can only work if a system is able retrieve and connect previously stored information about an object. This required information is the input data, which typically consists of a digital representation (image) of the object itself. An interesting area of computer vision is object tracking. In this discipline, algorithms try to recognize objects in consecutive video frames, and thus, to keep track of the object's location trough the total period of a video sequence. Some algorithms are able to expand their knowledge during the computation and therefore improve their performance with every processed frame.

In this thesis an application is designed and implemented which utilizes the cloud concepts of external computation in order to perform object tracking tasks requested by the user. Beside a proof of concept demo application, the core application can be seen as a service. This service can be integrated by developers in order to efficiently produce own cloud-based object tracking applications.

1.1 Problem Statement

Computer vision and especially object tracking are an active research area. There are plenty of different algorithms for solving the task of object tracking [1]. Some tracking algorithms are integrated in computer vision libraries like *OpenCV*[1] or *dlib*[2]. Other algorithms use common utilities of these libraries to process the frames of a video. These algorithms are mostly implemented in programming languages (*C++* or *Matlab*) suited for mainly developing offline desktop applications. Some algorithms are also implemented in Python [31]. This enables an easier implementation of object tracking server applications due to existing Python server libraries [32].

Server tracking applications can be utilized for the development of web-based tracking applications. The benefit of such applications is that the user does not

[1] http://opencv.org
[2] http://dlib.net

need to install any software but only requires an internet connection and a browser. Another advantage is the delegation of computationally intense tracking tasks from the user's machine to the server. Although available tracking algorithms offer methods to use and integrate their functionality into an application, the infrastructure of the server still needs to be implemented. A preferable solution would offer methods similar to the ones of the algorithms in order to integrate abstracted tracking functionality in the scope of a server application. Method-calls can be transmitted via an Internet protocol from the client to the server and the results back to the client. The developer can then focus again on building a tracking application, but with the ability to target a web-based platform with all its benefits.

There are already some related implementations addressing this problem. Wirewax[3] offers an online tool for making videos interactive. The user uploads his footage and is able to select objects in a frame. The location of these objects is than tracked through a section or the entire footage automatically. The user is able to define actions that are triggered when a consumer of the video clicks on the tracked object while watching the video. This tool can be used for creating interactive shopping clips where the consumer can put advertised items directly into the shopping card. Wirewax demonstrates the implementation of a web-based tracking service. The core limitation of the product is, that their tracking tool is bound to their system. The tracked data cannot be used outside the tool and the interactive videos and mechanics are hosted by them even when the user wants to embed the interactive video on his own website.

CloudCV[4] is a cloud service for delegating heavy computer vision tasks from a user's machine to their servers. The service offers image stitching, training and classification of images, object detection, and some other tools. CloudCV offers a *Python*[5] and *Matlab*[6] application programming interface (API) for integrating their service into own applications. A web interface can also be used to test the functionality via the CloudCV website. The computation on the servers supports

[3]http://wirewax.com
[4]http://cloudcv.org
[5]https://python.org
[6]https://mathworks.com/products/matlab

CUDA[7], a technology for parallel computation using Grafic Processor Units (GPUs). CloudCV demonstrates the implementation of a web-based service for computer vision tasks, that provides the necessary public API for integrating their service in order to build own applications on top of it. The API supports only backend programming languages. Therefore, in order to use the service in a web-based application, an additional backend application is needed. The main disadvantage of CloudCV is that it does not provide a tool for tracking objects in videos or consecutive frames. Only the related computer vision task, object detection, is supported. The user uploads an image with objects and the object detection tries to detect and classify these objects with the help of a training database.

1.2 Motivation

"The scientific man does not aim at an immediate result. He does not expect that his advanced ideas will be readily taken up. His work is like that of the planter — for the future. His duty is to lay the foundation for those who are to come, and point the way. He lives and labors and hopes." - Nicola Tesla (1934) [42]

Find an interesting and insufficiently resolved problem and try to solve it; distribute the results in a way that not only you will benefit from the solution but also others who may encounter the same problem; reuse available tools to build new reusable tools.
This is my understanding of contribution, especially in terms of software engineering. This statement is applicable to the problem stated in the previous section, given that there is currently no software available which can be used to support the development of cloud-based object tracking applications. The technologies such as tracking algorithms or transport protocols are available and they need to be combined together in a smart way. The goal is to take this process off the developers so they can focus on other parts of their applications.

[7] https://developer.nvidia.com/cuda-zone

1.3 Aim of the Work

The aim of this work is to analyze existing methods and technologies that are suited to implement a cloud-based object tracking service. These methods are then combined to develop a functional prototype. Finally, an evaluation points out the best performing implemented tracking algorithm. In general, this service allows developers to build cloud-based object tracking applications using an API provided by the service. The API can be used to delegate object tracking tasks from the client's machine to dedicated servers running the service. The tracking service should be able to track a single unknown object in a video. The user only locates the object in the first frame.

1.4 Outline of the Work

This thesis is structured as follows. Chapter 2 provides background information on the functionality and challenges of object tracking and an overview of the different cloud system concepts and the benefits of cloud computing. Chapter 3 addresses some state of the art methods for optimizing object tracking tasks. Chapter 4 analyses the methods used to implement the tracking service. First, requirements for the tracking service are specified. Next, the chapter provides information on the functionality of the tracking algorithms implemented in the service. Additionally, suitable protocols for transferring data between components are discussed. Chapter 5 documents the structure and the implementation of the tracking service. Chapter 6 evaluates the performance of the implemented tracking algorithms in context of low resolution footage or skipping of several frames. The last chapter provides a summary and addresses open topics for further work.

CHAPTER 2

Background

This chapter provides background information on central concepts of this thesis. The first section of the chapter gives an overview of the basics on object tracking, its use cases and challenges, and the different types of tracking algorithms. The second section provides information about cloud computing and its various service and deployment models.

2.1 Object Tracking

Object tracking is a major task in the computer vision domain. The aim of the task is to track one or more objects of interest over a sequence of frames in order to retrieve information about the spatial and temporal changes of these objects. There are multiple possible use cases for object tracking, for example [49]:

- **Traffic monitoring**: Track and count vehicles to collect traffic statistics.
- **Human-computer interaction**: Gesture recognition and eye gaze tracking as input for computers.
- **Vehicle navigation**: Recognizing obstacles in order to prevent accidents.
- **Automated surveillance**: Monitoring of scenes to detect suspicious activities.

- **Video indexing**: Annotation of footage in multimedia databases.

2.1.1 Challenges of Object Tracking

In order to successfully track an object, it has to be uniquely distinguished from the rest of the content in a paricular frame. This can be a challenging task due to scenarios that may occur in the footage, such as [39]:

- **Illumination changes**: Sudden illumination changes can occur in the scene due to changes in the environment lighting and cause false positive information.
- **Dynamic background**: Moving parts in the background of the scene such as clouds or other objects can distract the tracking process and may also lead to false positive results.
- **Occlusion**: The object of interest may move partially or fully behind other objects. The challenge is to recognize a partially occluded object of interest or to redetect the object after it was fully occluded for a certain amount of time.
- **Video noise**: If the footage is superimposed with noise, it is challenging to extract the object of interest from the background.
- **Camouflage**: When objects differ poorly from the background or other objects, tracking systems have difficulties to distinguish the correct object of interest.
- **Motion of the camera**: The video may be captured with unstable or unpredictable movement. This scenario may lead to loosing track of the object of interest.

2.1.2 Components of a Tracking System

Figure 2.1 shows an overview of the common components of a tracking system. In the first step, the system takes consecutive frames as input and extracts relevant

information from a predefined area where the object of interest is located (*feature extraction*). Different tracker use different approaches to identify the object of interest, such as motion classifications or extracting features (e.g., color, gradient, edges, and unique interest points).[23]

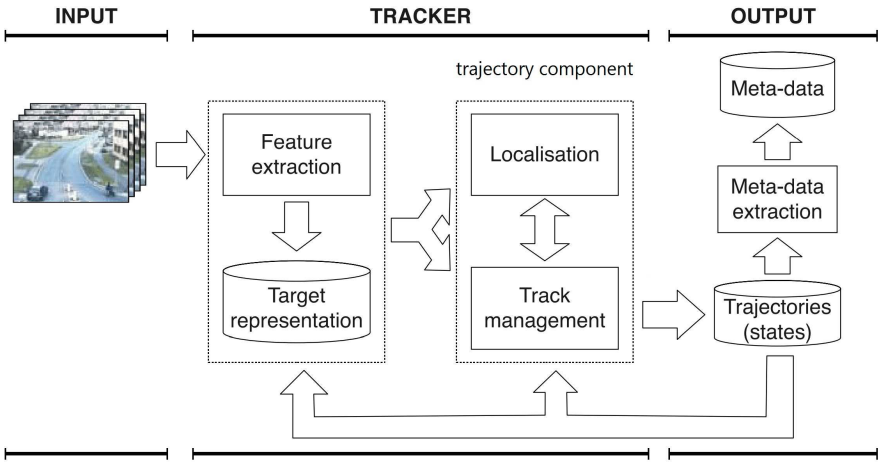


Figure 2.1: Main logical components of a tracking algorithm (figure adapted from [23])

In the next step the tracker creates a model based on the set of extracted features to represent the object of interest for further processing (*target representation*). The representation is a trade-off between the descriptiveness and the invariance of the object. If the object is described very precisely the probability for false positive results is minimized. However, the representation should be able to cope with changes in the target's scale, rotation, and occlusion.[23]

The *trajectory component* handles the object's appearing and disappearing from the scene. This component terminates the current trajectory if the object disappears and initializes a new trajectory if the target reappears. This is done by recursively estimating new trajectories based on the available states from previously tracked frames. Instances of the object are created and linked together to update the object model making it more robust. [23]

The *metadata extraction component* extracts meta-data from the current tracker state. The relevant data is application-specific and can be represented, for example,

as simple spatial coordinates of the object's location or it can be more descriptive, for example, classifying the mood of a human facial expression. [23]

2.1.3 Types of Tracking Algorithms

Tracking algorithms can be categorized into three classes by means of the level of interaction between the user and the tracking application:

- **Manual**: Manual tracking is performed completely by the user. It is commonly used when high accuracy is needed. The user defines a bounding box around a target frame per frame. This method is time consuming and inefficient for large quantities of data, e.g., labeling media databases. The film industry is using this technique because of the precision. [37]
- **Automated**: Automated tracking does not need any interaction with the user. An object detector can be used to detect and localize an object of interest automatically. This type of tracking usually needs a training database and is tight to a specific target. An example use case is to automatically detect moving objects in a scene for surveillance purposes. [37]
- **Interactive**: Interactive tracking is a trade-off between the two types of trackers above. This method is used in applications where the user manually localizes the object of interest in the first frame by dragging a bounding box around the object. In the following, the system automatically tracks changes in the location in the consecutive frames. Some trackers provide the option to manually redefine the bounding box if the system looses track. The core advantages of such interactive trackers are the flexibility in tracking unknown objects and the reduced computation time trough automation. [37]

Besides the categorization above, tracking algorithms can be grouped according to their strategy on how to track objects:

- **Point tracking**: The object of interest is represented by multiple points. Based on previous positions and motion, these points are associated to a

group. The object is detected in each frame by comparing the new points with old states. [49]

- **Kernel tracking**: The object of interest is represented by a rectangular or elliptical shape. The tracking is performed by tracking the motion of the shape between consecutive frames. [49]

- **Silhouette tracking**: The object of interest is represented by a combination of geometrical shapes. This type of tracker searches for the object's silhouette in each frame by comparing the contour of objects in the scene with the object of interest. [49]

2.2 Cloud Computing

Cloud computing is a computing model which provides access to remote computing resources. Infrastructure providers manage resources such as CPU or storage and distribute them mostly via usage-based pricing models. Service providers rent these resources to offer services and applications to end users. The applications are accesable over the internet [50].

2.2.1 Service Models

Cloud-based services can be categorized into three different models:

- **Software as a Service (SaaS)**: Providers give access to remote applications. The consumer has no control over the underlying cloud infrastructure but he can access these applications either trough a web page or via an API. A web page access is end user-oriented, for example, a web-based email application. The API access is developer-oriented. A developer can use an API to integrate a remote service to his own application. [25]

- **Platform as a Service (PaaS)**: Providers give access to an environment for developing, deploying, and operating applications. PaaS targets developers as main consumers. The user has the ability to push new versions of his

own application to the plattform. The code is automatically tested and deployed to a predefined configuration of depending software and hardware. Running applications can be monitored and managed via a web interface. The automation and abstraction of deployment leads to the possibility of developing applications rapidly [18]. Popular PaaS providers are Google App Engine[1], Heroku[2], Amazon AWS[3], and Windows Azure[4].

- **Infrastructure as a Service (IaaS)**: Providers give access to services for managing hardware to provide data storage, communication, and computing power. The user can orchestrate different combinations of hardware to build a customized scalable computing environment [35]. Popular IaaS providers are Google Compute Engine[5], Amazon AWS, and IBM Cloud[6].

2.2.2 Deployment Models

There are five major types of clouds for the deployment of applications:

- **Private clouds**: This type of cloud infrastructure is used exclusively by a single organization. The private cloud may be built and managed by the organization itself or by external providers. On the one hand, this solution offers high control over performance, security, and reliability. On the other hand, it is expensive and the setup is close to traditional sever farms. [50]

- **Public clouds**: In this type of cloud the infrastructure is open for the public. The management and operation of this system can be provided by the application provider itself or by a business, academic, or government organization [25]. The advantages of public clouds are the low costs and the delegation of possible risks to the providers. However, public clouds do not offer a high level of control over hardware resources and security. [50]

[1] https://cloud.google.com/appengine
[2] https://www.heroku.com
[3] https://aws.amazon.com
[4] https://azure.microsoft.com
[5] https://cloud.google.com/compute
[6] https://www.ibm.com/cloud-computing

- **Hybrid clouds**: This model is a combination of a private and a public cloud. Parts of the service that are not vulnerable to security issues run on the public and the remaining parts run on the private cloud. This composition offers relatively low costs while still having control over important parts in term of security and hardware. [50]
- **Community clouds**: Organizations with similar concerns (e.g., security, policy, mission) can share a community cloud infrastructure. This model is a generalization of a typical private cloud because more than one organization have access to it. In this variant the costs are reduced in comparison to the private cloud. [35]
- **Virtual private clouds (VPC)**: VPC is an encapsulated platform running on the top of a public cloud. Service providers can use the virtual private network (VPN) technology to configure own security settings (e.g., firewall rules). This type of cloud is low at costs but with remaining security aspects such as sandboxing. [50]

2.2.3 Benefits of Cloud Computing

There are two groups of users who benefit from the use of cloud computing in different manners. The first group are the service providers. Service providers can save costs in terms of used resources. A business does not need to pay for an own server. In the cloud domain only the used remote resources are payed. The resources can be scaled on demand if the service grows on popularity and more end users are using it. Another aspect is the reliability. Cloud providers are able to switch applications in run time to other available servers if a server crashes. Service providers outsource these concerns and are able to focus on more relevant parts of their business.

The other group are the end consumer of web-based applications in the cloud. Such applications are accessible trough the web browser so that there is no need to install additional software on the owner device. Additionally, the use of cloud applications are platform-independent. Applications in the cloud use remotely-located computation power. The user's computational resources are not needed

and can be used elsewhere. Moreover, a better hardware (located in the cloud) is able to compute tasks much faster.

CHAPTER 3

State of the Art

This chapter presents state of the art techniques used to improve and optimize the process of localization of an object of interest in a frame of a video.

3.1 Prediction Methods

Brute force methods for finding the position of an object of interest could take too long because the method alaways scans the whole frame. If the object is for example in the bottom right corner and the brute force search starts on the upper left corner, it would be very inefficient. This leads to an unnecessary overhead of computation time. Modern tracking algorithms assume the probable position (hereafter referred to as state) by taking the state of the previous consecutive frames into account. As a result the scanning window can be shrunk around the probable location. Only if the tracker does not find the object in the predicted scanning window, the tracker will start a complete new scan. Due to scene cuts it is possible that the object changes its location abruptly. There are three common methods to predict the object's position: [13]

- **Motion Model**: The motion model predicts the next state by taking the last state's velocity, direction, and acceleration into account [13].

- **Kalman Filter**: The Kalman Filter is used to predict object states in noisy environments when the object of interest moves in a linear manner. In a prediction step the algorithm estimates the state of an object at a specific time by taking previous state variables into account, e.g., by using the Motion Model. In a correction step the system updates the ranking of the previous estimations with a weighted average. Estimations that are more contemporary, are rated higher . [46]
- **Particle Filter**: Particle Filters use data such as color, motion, or sound to compute a density function out of samples from previous frames [27].

3.2 Segmentation of Objects

Segmentation is used to find meaningful regions in the scene. These regions tend to be possible candidates for the object of interest. The process of tracking is expensive, so that tracking algorithms use segmentation methods only on objects that are moving in the scene. The background subtraction method is an approach for extracting moving objects from the static background of the scene. The algorithm compares a frame with a reference frame. The differences of the two frames indicate moving objects. The reference image can be the previous frame or the median value of pixels from multiple previous frames. In cases of a static video footage without any camera movement the reference frame can by easily obtained by using an empty scene without any moving objects (see Figure 3.1). This method was proposed by Lo an Valestin. [19]

The removal of shadows improves the accuracy of the segmentation process. Shadows move with their objects and are, thus, identified as a meaningful segment by an object tracker. These false segments may than be recognized as a part of the object and lead to false information. To remove shadows a technique proposed in [7] can be used. The frame is converted into the HSV color space to analyze each pixel's color properties. Regions with shadows tend to have a characteristic reduction of saturation and brightness, while the value of hue remains the same. If pixels have these changes of properties they are identified as shadows. The rate of reduction is deducted by experiments [8].

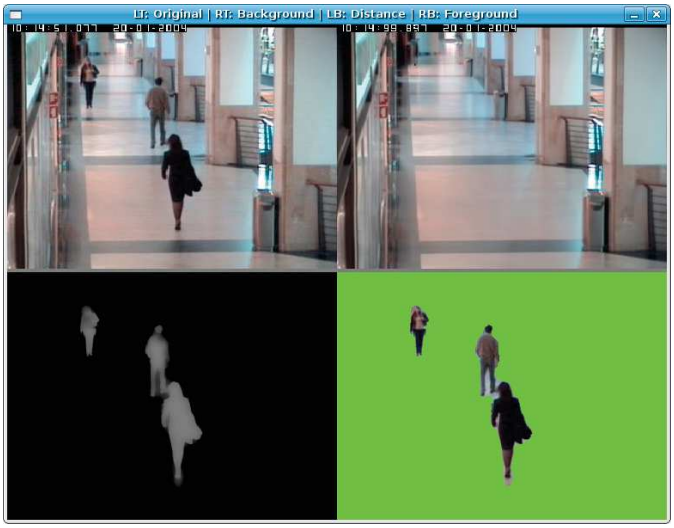


Figure 3.1: Background segmentation (figure from [38])

CHAPTER 4

Aproach and Methodology

The first section in this chapter deals with the requirements specification for the cloud-based object tracking web service implemented as a result of this thesis. The next section provides a summary of the functionality of three different tracking algorithms. These algorithms are suitable for solving the problem of tracking unknown objects over a longer distance in time. In the next section WebSocket is introduced. WebSocket is a communication protocol for web applications which provides a client server communication suitable for time consuming requests and bidirectional message exchange. The next section describes the advanced message queuing protocol (AMQP). This protocol provides communication between different clients on the server side and can be used to delegate tasks to multiple worker applications. The last section addresses the containerization of applications using Docker in order to make single parts of the application more flexible and easier to install.

4.1 Requirements Specification

The requirements for the cloud-based tracking web service can be categorized in three groups with respect to the different scopes of functionality . The first group are the requirements for a web service. A web service should run on a distributed system like a server so the service can handle multiple requests from users. The

server should open ports that are routed to the web service, so that users are able to access it remotely over the Internet. To enable the integration into applications, an application programming interface (API) should be exposed by the web service. The implementation of the server and web service is described in Chapter 5. The communication between the user and the service should use a common and suitable protocol to guarantee compatibility and offer durability for time consuming requests. For this case, the WebSocket protocol is analyzed in contrast to the classic HTTP protocol later in this chapter.

The second group specifies the requirements of a cloud-based web-service. A cloud-based web service should outsource computation intensive tasks from the users' hardware to a distributed hardware. In the scenario of this thesis, the web service delegates users' tracking requests to the hardware of the server. A cloud-based web service should be able to scale up, to handle more requests in parallel. To fulfill this requirement, Section 4.4 introduces The Advanced Message Queueing Protocol (AMQP). AMQP provides a solution to load balance tasks among multiple worker instances. Docker (presented in Section 4.5) offers additionally a simple way to spin up new worker instances on demand and to setup the service on common cloud platforms.

The last group specifies the requirements for the functionality of the service. The service should be able to perform the tracking of an object in a specific video footage. The location coordinates of the object of interest in the first frame should be provided by the user by attaching parameter to the request. The user should also be able to upload the footage to the server, so that the tracking can be performed on the desired footage. Algorithms need to be implemented, to perform the actual tracking task. The next section introduces three different tracking algorithms, which are suitable for tracking unknown objects when having an initial position as input.

4.2 Summary of tracking algorithms used

This section provides summaries of the algorithms' functionality, which are implemented in the tracking service. The three algorithms are selected, due to their

popularity and usage in the computer vision community. The first algorithm is named Tracking-Learning-Detection (TLD). The developer of this algorithm *Zdenek Kalal* was rewarded with the *UK ICT Pioneers* award in 2011. Additionally, the state of the art library for computer vision, `OpenCV`[1], has added this algorithm into its feature set. The second algorithm, *Clustering of Static-Adaptive Correspondences for Deformable Object Tracking (CMT)*, received the *Best Paper Award* at the *Winter Conference on Applications of Computer Vision*[2] in 2014. *Georg Nebahay*, the developer of CMT, also created a port of the TLD algorithm from Matlab to C++. The project page of the port[3] refers to the more novel algorithm CMT, which is one more reason why CMT attracted attention and is in the pool of the algorithms of this thesis. The last algorithm is a correlation tracker, which is implemented in the well-established machine learning toolkit `dlib`[4]. The original algorithm of the dlib implementation (DDST) was one of the best performing algorithms in the VOT challenge 2014[5].

4.2.1 Tracking-Learning-Detection (TLD)

The Tracking-Learning-Detection algorithm consists of three components which work simultaneously (see Figure 4.1). The tracker component follows the object of interest frame by frame. The detector localizes all appearances that have been observed in the past frames and corrects the tracker if any errors like drifting occur. The learning component considers the errors of the detector and updates the detector to avoid the same errors in the future. [15]

Tracker

The Median Flow Tracker [14] is used in the TLD framework. It uses a novel approach to overcome the problem where drastic changes in the appearance of an object or its disappearing from the viewport lead to failures in the tracking. The tracker tracks a single keypoint in consecutive frames. The location of the point in

[1](http://opencv.com)
[2](http://www.wacv14.org)
[3](http://gnebehay.com/tld)
[4](http://dlib.net/)
[5](http://votchallenge.net/vot2014/results.html)

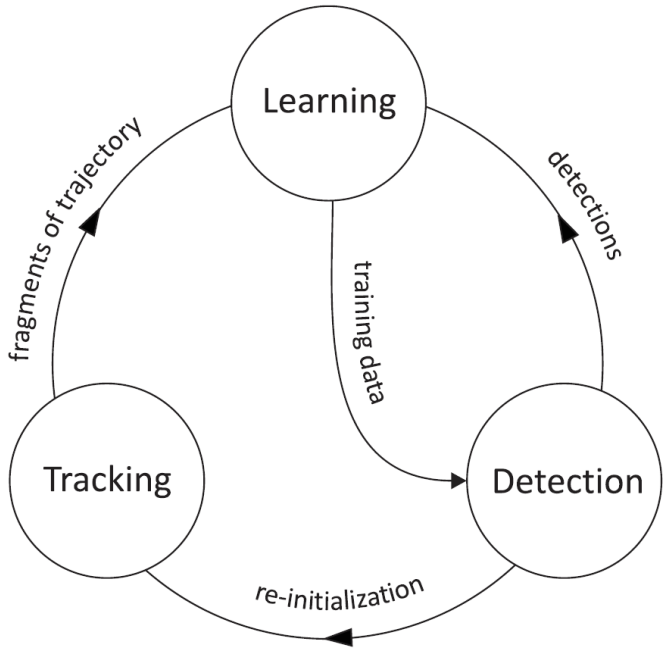


Figure 4.1: The block diagram of the TLD framework (figure from [15])

the last frame is validated by a backward tracking to the previous frame again. If both generated trajectories differ more than a threshold value, the first trajectory is considered to be incorrect (see Figure 4.2). [15]

To initialize the tracker, usually, characteristic keypoints (e.g., localized on prominent, dominant edges) of the first frame are detected [36, 41]. To overcome the problem that keypoints could disappear or become occluded and in this way leading the tracker to fail, a brute force mechanism is implemented that tracks all pixels of the first frame through the whole footage. The created trajectories are than evaluated by the forward-backward error detection. A resulting error map of every point shows their reliability trough the whole sequence.[15]

An equaly spaced set of points in the bounding box of the object of interest is constructed. The points are tracked by the Lucas-kanade tracker to generate a sparce motion flow between two pairs of frames. A quality estimation of the point's position prediction is made with the forward-backward technique. 50% of the worst predictions are removed. The remaining points provide the new bounding box (see Figure 4.3). [22]

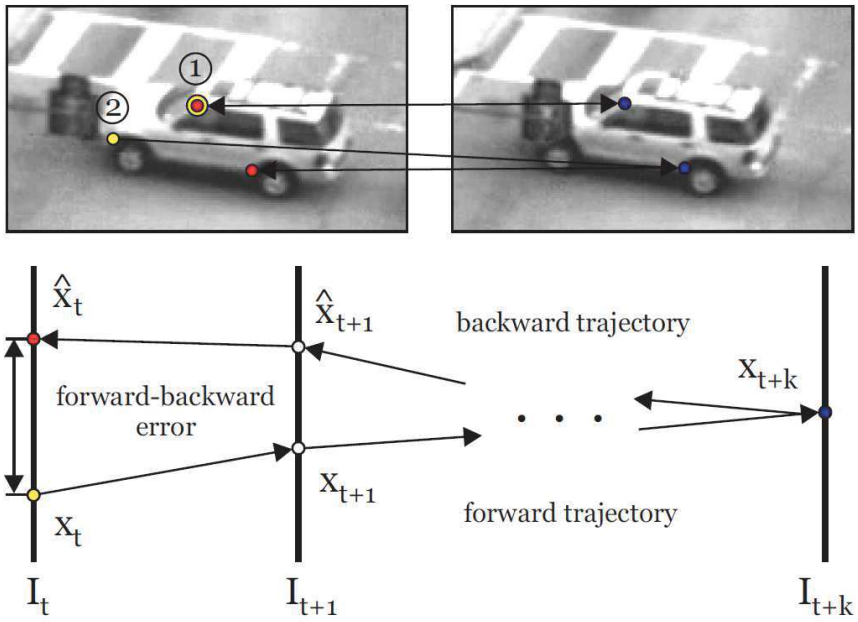


Figure 4.2: Forward-backward error detection. Point 1 leads to a correct result, whereas point 2 is a mismatch (figure from [14])

.

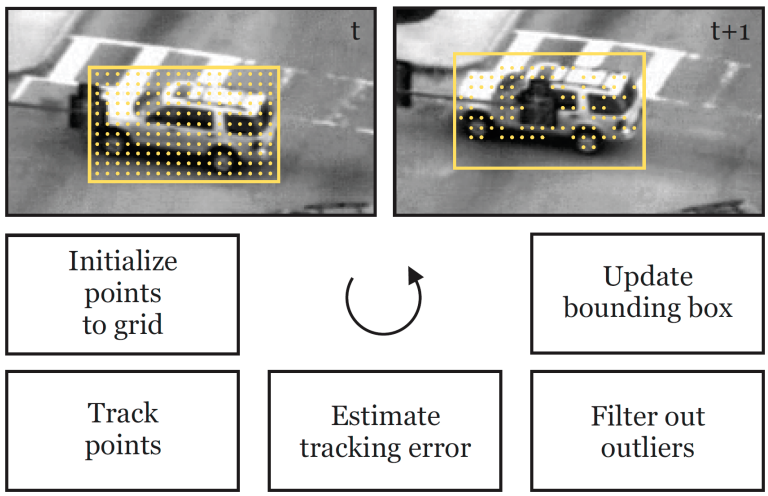


Figure 4.3: Block diagramm of median flow tracker (figure from [14])

Detection

Each frame is scanned by a scanning-window to determine in which patches the object is present. In following, a variety of different angles shifts and scales of the

initial bounding box is created. This leads to many thousands possible bounding boxes to be evaluated. To speed up the process a cascaded classifier is used to reject many bounding boxes in early stages of the cascade. [15]

The first stage is called patch variance. All patches get rejected if the gray-value variance is smaller than 50% of the object of interest. In this stage, typically more than 50% of the patches are eliminated. [15]

The next stage is the ensemble classifier. Multiple base classifiers perform pixel-based density comparisons of the remaining patches. For each test the probability is calculated if the patch contains a part of the object of interest. The algorithm rejects a patch if the probability is less than 50%. This method is not as fast as the first stage. However, it is still faster than other common classification methods using local features such as SIFT, as evaluated in [30]. [15]

In the last stage only a few bounding boxes are left. A nearest neighbor classifier is used to determine the patches where the object is located. A patch is classified as correct if its relative similarity to the object model from the learning component is greater than a threshold value that was empirically determined in the range between 0.6 and 0.7. If too many positive patches are stored then some randomly selected ones are sorted out. The threshold for the amount of patches that can be stored is limited by the given computer memory but several hundred templates can be stored. [15]

Learning

Due to lack of a dataset at the beginning, in the first frame the learning component trains the detector using generated labeled examples. To generate the first set, 10 bounding boxes are selected which are close to the object of interest. For these bounding boxes 20 different geometrical transformations in scale, shift, and rotation are created. Negative examples are gathered from the surroundings. With this initial dataset the object model can be built and is ready to get updated by data processed in the following frames. [15]

To update the detector two different so-called experts are used. The P-expert discovers new appearances, whereas the N-expert is used to generate negative

training examples. The P-expert identifies reliable parts of the tracker and generates new training examples using geometrical transformations on the fly. The N-expert gathers negative examples by using previous data of the tracker, that was not identified as the object of interest. [15]

4.2.2 Consensus-based Matching and Tracking of Keypoints for Object Tracking (CMT)

CMT uses a bounding box to define the object of interest in the first frame. In the selected region multiple keypoints are initiated and their coordinates are mean-normalized. In every frame the goal is to identify a group of matches that represents the object. [29]

Static Adaptive Correspondences

A static appearance-based model is build on the basis of the location of the object of interest in the first frame. Matches of this model are referenced as static correspondences. As the time gap between the current and the first frame gets unpredictable large, purely appearance-based methods has to be applied in order to connect the correspondences. [29]

A global search is used for the detection and establisment of matches between keypoints of the first frame and possible candidates in the active frame. This is done using a threshold and an employment of the second nearest neighbor distance criterion [20] on the distance between points descriptors. Additionally, candidates are excluded that are matching with the background descriptor applied on the first frame. The static model is able to redetect keypoints that moved out of the viewport and updates itself to reemploy those missing keypoints if they reappear. [29]

The adaptive model gets updated every frame and takes image patches around the keypoints of the last frame's match into account. Due to the minor changes between two consecutive frames no global search is needed. By using optical flow a correspondence can be created using fast local optimization [22]. To filter

out false correspondences a forward-backward error measure, the same as in the TLD-algorithm, is employed [14]. [29]

The adaptive correspondences are overruled by the static ones because the static correspondences do not have drift errors that gets eliminated by the error measurement. The combined correspondences are referred as L_t^*. [29]

Correspondence Clustering

To represent the deformation of the object of interest, an estimated similarity transform H from L_t^* is used to calculate the dissimilarity measure D between two matches m_i and m_j (x^0 is the location of a keypoint in the initial frame and x^t is the location of a keypoint in the current frame):

$$D(m_i, m_j) = \left\| (x_i^t - Hx_i^0) - (x_j^t - Hx_j^0) \right\| \tag{4.1}$$

As shown in Figure 4.4 the initial keypoints are transformed into the coordinate system of the current frame in order to calculate D. L_t^* is then divided into subsets using an agglomerative clustering algorithm [48] with the similarity measure, H, as input. This results in two-dimensional clusters with a cutoff threshold δ to set the degree of possible deformation (see Figure 4.5). [29]

The biggest cluster L_t^+ is considered as the one which contains the correspondences relevant for the object. The rest of the clusters are disturbances. For the reconstruction of the current bounding box in dependence of scale s and rotation α, heuristics [14, 28] are used to estimate the values with respect to the initial state of the corresponding keypoint pairs (*med* denotes median): [29]

$$s = med\left(\left\{ \frac{\left\| x_i^t - x_j^t \right\|}{\left\| x_i^0 - x_j^0 \right\|}, i \neq j \right\}\right) \tag{4.2}$$

$$\alpha = med\left(\left\{ atan2(x_i^0 - x_j^0) - atan2(x_i^t - x_j^t), i \neq j \right\}\right) \tag{4.3}$$

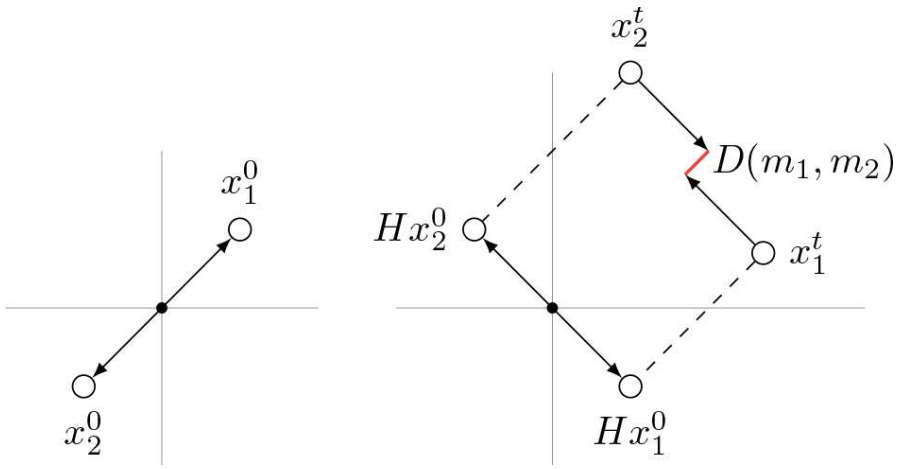


Figure 4.4: Computation of dissimilarity measure D using similarity transformation H (figure from [29])

.

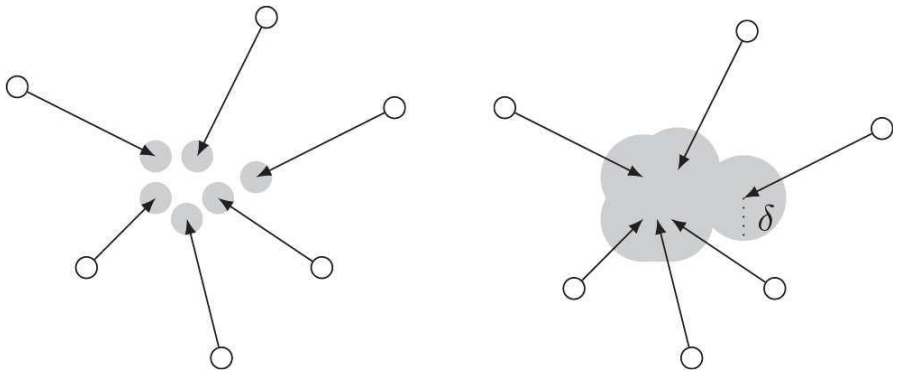


Figure 4.5: Left: δ is small and keypoints are not recognized as one cluster. Right: δ has the optimal value to form a cluster (figure from [29]).

Disambiguation of Correspondences

By disambiguating correspondences, the problem of similar descriptors appearing on the object can be handled. As shown in Figure 4.6 candidate keypoints are excluded if they are geometrically different to L_t^+. By excluding those keypoints the output can be computed with a higher accuracy and the matching set for the adaptive correspondences can be tuned for the following frames. [29]

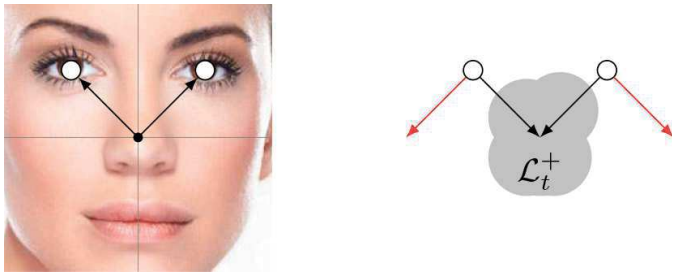


Figure 4.6: Similar keypoints will become outliers, due to different geometric properties with L_t^+ (figure from [29]).

4.2.3 Correlation Tracker

This subsection gives an overview of the employed tracking using a correlation filter called Minimum Output Sum of Squared Error (MOSSE). Creating filters of an object of interest can lead to strong peaks of the object but also can respond falsely to the background (see the naive filter in Figure 4.7). MOSSE produces more robust filters when the appearance changes. Additionally, the differentiation between background and the object of interest is more robust by applying preprocessing steps on the pixels explained below. Average of Synthetic Exact Filters (ASEF) [4] and Unconstrained Minimum Average Correlation Energy (UMACE) [24] are two approaches for the detection and identification of objects of interest. MOSSE is an adaptation of these techniques to enable online training and the possibility to update the filter on the fly. These additional features allow for a robust visual tracking, instead of only using the filter for detection and identification. [3]

The object of interest is selected manually by a bounding box in the first frame. MOSSE uses example images of the object of interest to train an appearance model of the object. The initial training set consists of eight unique affine transformations constructed from the bounding box area. They represent the object of interest in several appearances with different rotations and scaling. In the following frames the filter training and the tracking work together. [3]

To overcome potential problems with low contrast lighting scenes, the pixel values are transformed using a log function in a preprocessing step. In the next step, the pixel values are normalized. The image is then multiplied with a cosine filter

to gradually reduce the pixel values on the edges. The cosine filter puts more emphasis near the center of the object so that the background can be better distinguished. Figure 4.7 shows the performance of the MOSSE algorithm and its related algorithms ASEF and UMACE in comparison to a non-optimized naive filter. [3]

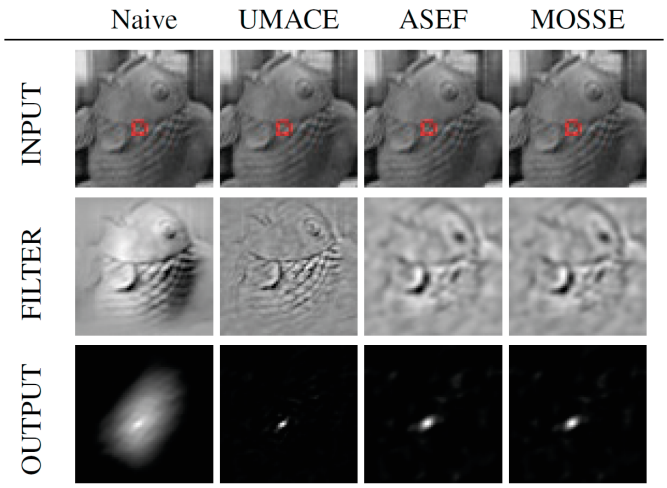


Figure 4.7: The output of the correlation-based filters is more precise than the output of a naive filter in terms of the isolation of the object of interest (figure from [3]).

The filter correlates over a sliding window in each frame. The location of the pixel with the maximum correlation value indicates the new position of the object. To speed up the computation process, the filter and the image are converted into the Fourier domain using the Fast Fourier Transform (FFT). The Convolution Theorem [2] states that a correlation operation becomes a multiplication in the Fourier domain and vice versa. After the computation in the Fourier domain, the result is converted back into the spacial domain using the inverse FFT. The filter is trained by constantly adding new results from previously processed frames to the training dataset. To quickly adapt changes in the scene, the more recent frames are considered with a higher weight to train the filter. [3]

A peak strength measurement can be done by calculating the Peak to Sidelobe Ratio (PSR) in order to estimate if the tracking failed or if the object is out of the scene. To measure the PSR, the correlation output is split into the peak (the

maximum value) and the sidelobe (the rest of the pixels outside of an 11×11 window around the peak). The PSR is calculated by subtracting the mean of the sidelobe from the peak and dividing the result by the standard deviation of the sidelobe. A range between 20 and 60 PSR indicates a very strong peak. When the PSR drops to less than 7, the object is probably occluded or the tracking failed. In this case, the algorithm disregards these outputs in order to avoid training using incorrect data. [3]

4.3 WebSocket

WebSocket is a full-duplex communication protocol between a client and server. WebSocket was standardized within HTML5 in 2011 but has an implementation in any common programming language [6]. An open TCP connection allows for a high frequent message exchange with a small overhead in real-time. Possible use cases are chat applications, stock market websites, and multiplayer browser games.

4.3.1 WebSocket vs. HTTP

To realize a real-time behavior in web applications, different strategies can be used. Using the HTTP protocol the client sends an HTTP request to the server in order to receive a response. This is sufficient for static pages. For frequently changing data in web applications like chats, the client would constantly need to send requests to server to receive the updated content. This technique is called polling and was the first attempt to deliver real-time information to the browser. This strategy is not effective, when the server does not have any updated content. Useless requests are initiated and lead to slowing down the application by flooding the network. [21]

A technique called long-polling minimizes the problem of frequent requests by sending an open request to the server. The server keeps the request open for a predefined period of time. If a new update is available during this period, the server sends the responce with the update to the client. The server terminates

the request if nothing was updated. There are no benefits in comparison to the traditional polling if an application has highly frequent updates. [21]

Another strategy is called streaming. The client sends a request to the server. The server returns an open response which is continuously updated with possible new data. Proxy servers may buffer these updates resulting in an increase of the latency to deliver messages. A TLS connection can be used to overcome the buffering, however, it has an increased demand on hardware resources. [21]

WebSocket offers some advantages in comparison to the HTTP strategies mentioned above. WebSocket establishes only one bidirectional connection between a client and a server. By sending data trough an open WebSocket connection, no HTTP headers need to be send leading to a smaller overhead. The small size of messages makes WebSocket preferable for high frequent message applications. The bidirectional connection allows to push messages to clients at any time. WebSocket is integrated into HTML5 in all modern browsers. This leads to high compatibility without any additional libraries. The WebSocket API offers a simple connection to the server. Messages can be received by triggering events, leading to non-blocking code. [21]

4.3.2 The WebSocket Protocol

A new WebSocket connection between a client and the server begins with an HTTP request acting as a handshake mechanism. The request has an upgrade header attribute assigned to indicate an upgrade of the connection to the WebSocket protocol. To complete the handshake, the server has to respond with a computed key to verify its capabilities of the WebSocket protocol. To compute the key returned as the Sec-WebSocket-Accept header, the server takes the value of the client's Sec-WebSocket-Key header and decodes it. After a successful handshake the connection uses the TCP protocol instead of HTTP to exchange data. [45]

The clients and the server can send messages to each other while the connection is open. A message, also referenced as a frame, is transmitted within the header in binary format. The frame consists of different code parts with specific responsibilities (see Figure 4.8). [45]

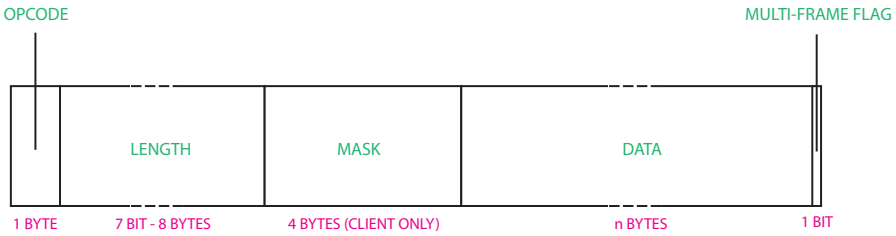


Figure 4.8: Coding of the frame header (adapted figure from [45])

The first four bits are representing an opcode indicating the type of the message payload. The payload can be a message transmitted as text or binary, a closing handshake, or a ping/pong message to check if the other side is still available. Other possible combinations of the bits are reserved for possible future types. The following bytes encode the length of the payload. According to the length more or less bytes are needed to represent the value. This prevents unnecessary bytes in small messages. Frames from clients to the server are masked using the next four bytes. This is mandatory to prevent cross-protocol attacks. Such attacks redirect the traffic of one protocol to another to spoof commands [40]. The payload block is variable according to the length and is either binary or encoded in 8-Bit UCS Transformation Format (UTF-8) for the text data type. A final bit indicates a multi-frame message. This allows streaming of data or sending parts of bigger messages separately. [45]

When one side wants to gracefully close the connection, it sends a closing handshake. With this handshake the opposite party can distinguish between an aborted closed connection due to network errors and an intentionally closed connection. The aborting side can additionally send a numerical code that gives a more meaningful reason for terminating the session. [45]

4.3.3 Use Case in a Tracking-Service

Tracking of objects in a video can be a time consuming task. However, a single open TCP connection of WebSocket is able to handle long time gaps between a job request and the response. Moreover, the response is not depending on a specific request. A request may lead to multiple responses. Every tracked frame provides

updated information about the location of the object of interest. This information can be pushed to the client in order to provide real-time status updates of the progress. The event-driven API on the client side allows in addition a non-blocking usage of the application which uses the tracking service.

4.4 Advanced Message Queuing Protocol (AMQP)

AMQP is a networking protocol which provides communication between different clients trough a middleware broker using messages. Its goal is to connect a wide range of different applications regardless of their structure. The protocol is defined trough a model which consists of components that route and store messages in a broker service (see Figure 4.9). [43]

The exchange component takes the messages from publishers and routes these according to rules defined in the binding component to the queue component. The message queue component saves the messages and processes them to the subscribed clients when they are ready to receive them. The clients have the ability to acknowledge that the message was received successfully. If the broker does not receive the acknowledgment of the client due to network errors the broker is able to both resend the message and contact the sender. [43]

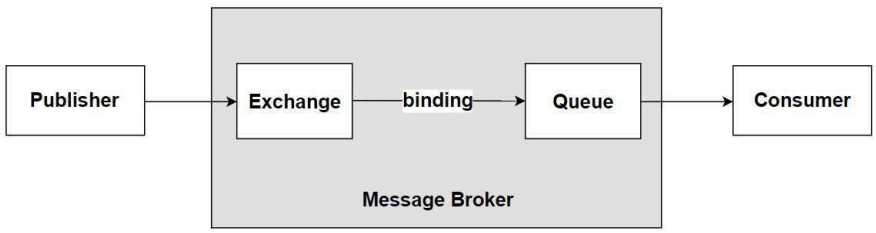


Figure 4.9: Advanced Message Queuing Protocol components.

4.4.1 Exchanges

AMQP provides different types of exchanges, which are described in the following.

Default Exchange

The default exchange is the simplest method to use AMQP. Every queue that is created gets automatically bound to the exchange by the queue's name [43].

Direct Exchange

A specific message routing key is defined to deliver messages from the direct exchange to the queues. This type of exchange is ideal for unicast routing to distribute tasks to multiple instances of a client referenced as worker. The algorithm works as follows. A queue binds to the exchange with a routing key K. When a new message with routing key R arrives, the exchange checks if K and R are the same and routes the message to the queue if the condition is true. The incoming messages are load balanced between the consumers if they are subscribed to same queues (see Figure 4.10). [43]

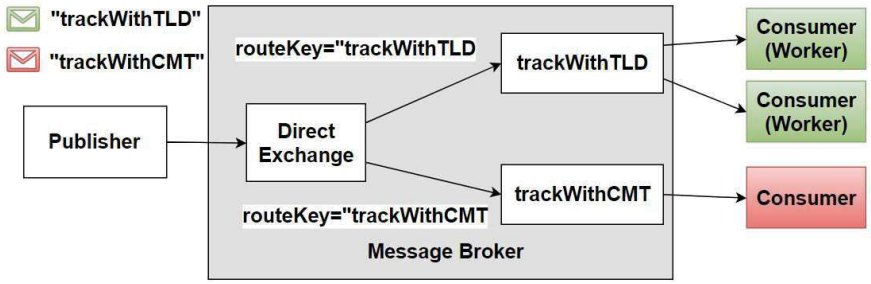


Figure 4.10: The message broker uses the direct exchange method to route specific messages to the corresponding queues.

Fanout Exchange

A fanout exchange ignores the routing key and routes incoming messages to all queues that are bound. No load balancing is used and all queues receive the message (see Figure 4.11). This type of exchange is used to broadcast messages to clients. It can be used to realize messaging in massively multi-player online games (MMO) or group chats. [43]

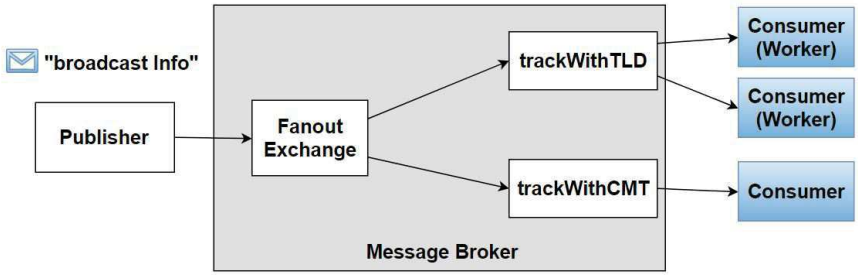


Figure 4.11: The message broker uses the fanout exchange method to broadcast a message to all queues.

Topic Exchange

The topic exchange matches a specific routing pattern of the queue with the message's routing key. The messages only get routed to the matching queues. Wildcards can be used to design a flexible routing where only a group of subscribers gets a portion of message types (see Figure 4.12). Topic exchanges are used to implement multicast routing of messages. [43]

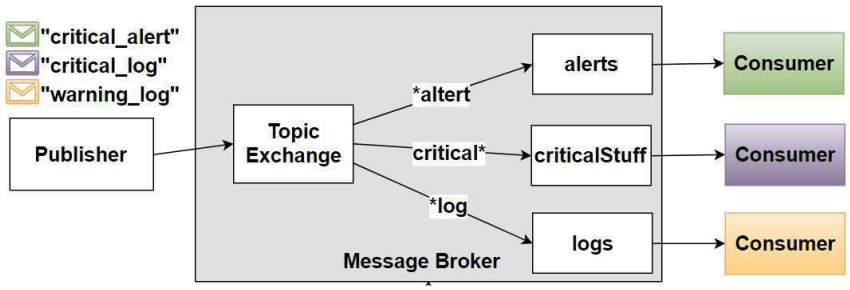


Figure 4.12: The message broker uses the topic exchange method to route messages to queues that matches a pattern.

Header Exchange

A header exchange does not consider a routing key for forwarding messages but it uses attributes of the message instead. If the value of an attribute in the header equals to a value specified in a table of arguments upon binding, the message gets routed (see Figure 4.13). A queue can be bound to more then one header attribute.

In this case a setting argument "x-match" needs to be specified to tell the exchange if all headers must match or if any of them is sufficient. [43]

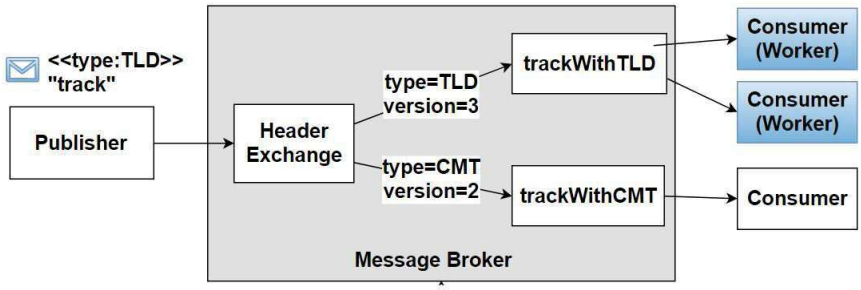


Figure 4.13: The message broker uses the header exchange method to route a message only to the queue with the correct type-attribute.

4.4.2 Queues

A message queue is a first in first out (FIFO) buffer that stores messages addressed to consumer applications. If more consumers are subscribed to a queue, different dispatching strategies can be used. The round-robin dispatching is the default technique to schedule messages in the queue. The messages are evenly distributed among the consumers. Fair dispatching in contrast assigns a message to a consumer if the consumer has acknowledged to be ready for the next message [17]. This way the tasks can be load balanced effectively (see Figure 4.14). If a worker receives a time consuming task, the queue can distribute the messages to free workers [33].

If a publisher sends a message, the message stays buffered in the queue until a consumer picks up the message. If the queue is declared as durable and the message as persistent, it is stored on the hard disk of the broker. If one of the specifications is not enabled, it is stored in the memory. [5]

4.4.3 Use Case in a Tracking-Service

AMQP can be used to implement worker applications which run a specific algorithm. Using the load balancing features, these workers can be scaled up on demand if more users are using the service. A queue management is able to address specific tasks

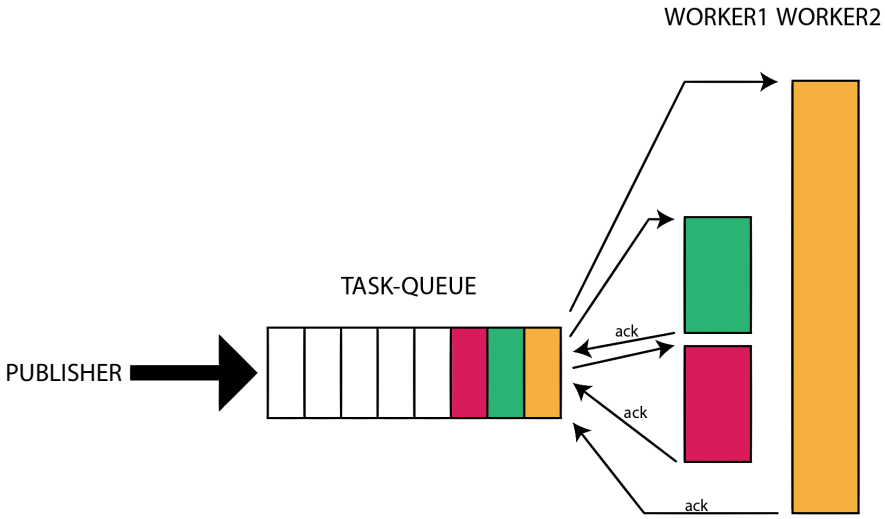


Figure 4.14: The publisher sends tasks to the queue. The tasks are distributed to two workers using fair dispatching.

to these workers. AMQP offers libraries for any popular programming language. The worker applications can be implemented in different programming languages with the benefit of using existing public implementations of the tracking algorithms. Additionally, more tracking approaches can be added in an efficient modular way later on.

4.5 Docker

Docker is a leading open-source engine for the deployment of server applications into containers. A container bundles applications with their dependencies in an isolated lightweight virtual environment. Created containers can be shared among other docker-driven systems. This leads to easy and failproofed installations on production servers. [12]

4.5.1 Advantages and Use Cases

Docker enhances consistency between the development environment and the production environment. An encapsulated Docker container can mimic a server setup.

The developer can test the code in this container. This reduces the risk of failure when running the software on the server later on. A even better approach is to run Docker on the production server itself. A Docker-driven server can run containers that were built on other systems and thus it can guarantee identical functionality. [44]

Docker is very lightweight for system resources, allowing to use Docker as a development tool on low performing desktop machines. In contrast to conventional virtual machines, Docker does not need guest operating systems to be installed. Multiple containers run on a single machine and share the same operating system kernel. Additionally, a layered file system shares common files between containers marking them more efficient.

Docker allows to develop microservice-oriented applications by following the concept of only one service per container [26]. This helps to update and manage single services. Containers can be linked to each other, providing the necessary infrastructure for microservices. Docker Compose offers easy tools for configuration and starting multi-container applications. Parts of an application such as workers can be scaled up with Docker. This leads to load balancing the service on demand [11].

4.5.2 Docker Architecture

Docker consists of several core components. The Docker daemon runs on a host machine and has the job to build, run, and distribute docker containers. A Docker-client sends orders to a local or remote Docker daemon (see Figure 4.15). [10]

Docker images are templates with instructions for building and running software packages. The instructions can be commands for running scripts, adding files, or installing new software. Each image is based on an existing base image (mostly a minimal linux distribution), which is then extended. The user has the option to create new base images which encourages reusability and abstraction. Docker uses a union file system to combine single steps of the templates as layers to a single image. This layering technique has the benefit to only update changed instructions and is, thus, fast when rebuilding the image. [10]

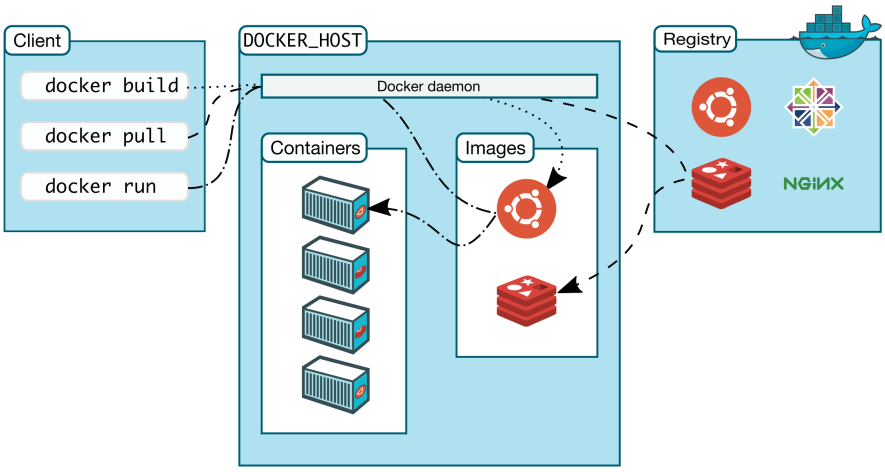


Figure 4.15: The core components of Docker (figure from [10])

A Docker-registry is a store for Docker-images. The constructed images can be shared on a public registry (Docker Hub) or on a local private registry. Docker Hub additionally offers a paid private registry. When running a Docker image, the base image gets pulled from the registries automatically. [10]

A Docker-container is an instance of an image and consists of the base operating system and additional user-added files declared in the template. A container is an isolated environment where an application can run. Containers are able to communicate with each other to share data. [10]

4.5.3 Use Case in a Tracking-Service

Docker simulates the production environment. This way the tracking service can be implemented and tested on a local machine. Moreover, many cloud providers offer a Docker integration, allowing to easily install the whole application in the cloud. New dedicated worker machines can be provisioned and started on any machine capable of Docker and connected to the running application.

CHAPTER 5

Implementation

This chapter documents the implementation of the tracking web service. The different components and their responsibilities are described in the following sections.

5.1 Overview

The application consists of multiple components (see Figure 5.1). The WebSocket server is the core of the application and connects all other components. Clients connect via WebSocket to the server and can use the exposed application programming interface (API) to upload video footage, to send requests for tracking, and to receive results. The client wrapper library allows developers to integrate the service by wrapping the API calls into easy to use methods. The tracking requests are routed from the WebSocket server trough a RabbitMQ[1] server to the worker applications. Each worker implements a specific object tracking algorithm. The service provider has the option to spin up multiple workers of the same type on the fly. The tasks are then load balanced between the workers.

[1] https://www.rabbitmq.com

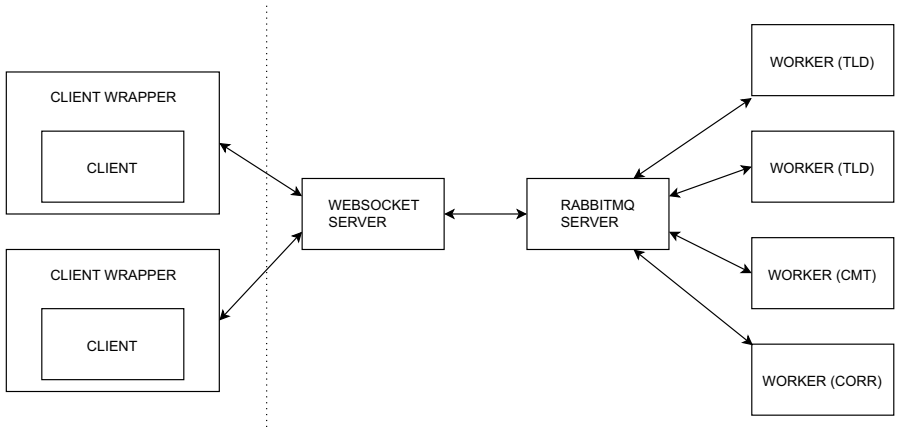


Figure 5.1: Components of the tracker service.

5.2 WebSocket Server

The WebSocket server is implemented as a Java application running on a GlassFish[2] server. The WebSocket server provides an API which can be used by clients in order to send tasks to the server. The server routes tracking tasks to worker applications, while performing upload tasks of the user's video footage by its own.

5.2.1 Handling Clients

Each time a new client connects to the server a new `SocketSession` is created, which holds the unique session id assigned by the server. The `SocketSession` object provides functionality for uploading, tracking, and emitting messages to the client. When a client sends an API call, the corresponding `SocketSession` is obtained from the `SessionStore` and the exact state of the client's running tasks like uploading or tracking is correctly represented.

5.2.2 WebSocket API

The annotation method `onMessage` is called every time a new socket message is received. The message is passed to the `messageRouter` method. The message router checks if an action attribute in the message equals to an implemented action

[2]https://javaee.github.io/glassfish/

on the server. If nothing matches, the client receives an error message. Otherwise the specific action is executed.

The server exposes three different actions for using the tracking service. To access them, the client needs to send a message containing a stringified JSON object with an action attribute representing the name of the action and a payload attribute with specific action data:

- `initUpload`: A new file upload is initiated with a specific file name and the file size as payload data. If the initialization was successful, the client is able to send byteBuffer-encoded chunks of data, which get reassembled to a complete file when the file size is reached.
- `initTracking`: This action allows to set an uploaded image of an object of interest as a reference image. This reference can be later assigned to a tracker which takes an image instead of coordinates as an initialization method. This action is currently not in use since none of the implemented algorithm is using a reference picture.
- `track`: This action starts a particular tracking algorithm specified by an algorithm attribute on an uploaded video file defined by a file name attribute. The client has to provide bounding box parameters and the corresponding time in milliseconds or the position in frames, where the object of interest is located at a specific moment. If no end attribute is provided, the tracking will be performed until the end of the footage is reached.

5.2.3 Emitting Data to Clients

The bidirectional connection of the WebSocket protocol between a client and a server offers a communication where the server can directly send messages to the client. The client does not need to make a request to ask if the server has updates. This can be used to employ real-time notifications for clients. If a client is uploading a file, the server is notifying him about the current progress or a possible error. Messages from server to client are also used to transmit results of each frame in real

time. The results consists of the corresponding time information and the updated spatial bounding box parameters.

5.3 Client-Wrapper

Every system capable of WebSockets[3] can connect to the server and use the tracking services. To simplify the use of the API in browser applications, a client wrapper written in JavaScript is distributed by the GlassFish server. This wrapper exposes the following methods that structure the communication with the WebSocket server.

- `open`: Establishes the connection to the server by providing the address as parameter.
- `emitAction`: Transmits an action with a payload to the server. This method is meant to provide the possibility to extend the wrapper if more API actions are exposed by the server.
- `uploadFile`: Is a combination of two `emitAction` calls and simplifies the process of initiating a file upload and uploading the file itself.
- `track`: Combines the optional initialization of a reference image and the tracking.

The wrapper is also able to listen to events. If the server emits an action, the corresponding event is triggered on the client's side:

- `open`: server established a connection.
- `close`: connection was closed by server.
- `error`: error thrown by server.
- `uploadStatus`: progress of current upload.

[3] http://caniuse.com/#feat=websockets

- `trackingStatus`: progress of current tracking (payload has the current coordinates).

- `trackingData`: tracking is finished (payload has all coordinates).

The client wrapper is an essential tool that allows developers to integrate the API of the tracking service into new applications. The code listing 5.1 demonstrates an example call of the wrapper to document the usage.

Listing 5.1: Functionality of wrapper methods

```
1 const s = new webSocketAPI();
2 s.open('ws://localhost:8080/webSocketAPI/actions').then(() => {
3     s.uploadFile(file, {
4         fileName: 'test.mkv',
5         chunkSize: 1024 * 1024 //1MB chunks for splitted uploading
6     }).then(() => {
7         s.track({
8             start: 3000, //start time in ms
9             end: 7000, //end time in ms
10            algorithm: 'tld', //other options: 'cmt' or 'corr'
11            fileName: 'test.mkv',
12            box: { //bounding box of desired object at given start
13                left: 100,
14                top: 200,
15                height: 400,
16                width: 500
17            }});
18    });
19 });
20
21 //triggered when new frame was tracked
22 s.on('trackingStatus', (message) => {
23     let mapping = JSON.parse(message);
24     //do stuff with data
25 });
```

5.4 RabbitMQ broker

RabbitMQ[4] is an open source message broker which uses the Advanced Message Queuing Protocol (AMQP) for communication. A client application can use a library available in any modern programming language to send and receive messages trough the broker to any other client.

Tracking can be a time and computation power consuming task. To unload the server, tracking jobs are passed to other worker applications. Using RabbitMQ as a message broker, the server acts as a publisher and sends jobs to the broker. The broker puts the jobs in a queue. Worker which are subscribed to a queue fetch these jobs in order to process them. One advantage of this setup and the abilities of RabbitMQ is, that worker applications can specialize to one particular job. Different workers can solve problems with different algorithms. This way the worker remains compact and better maintainable. The publisher sends specific tasks into queues that will be consumed only by workers, which are able to handle the job.

Another benefit is the system independent communication of AMQP. Clients can be written in nearly any language and run on any operating system. Each worker can be physically separated on different machines, connected trough a network. If more workers are subscribed to the same queue, the tasks get load balanced between them. New workers can be added on run-time, which allows the system to scale up on demand.

5.5 Workers

The tracking service offers three different worker applications implemented in Python. Each worker implements one of the three algorithms presented in Chapter 4 to process tracking tasks requested by the clients using the Websocket API. The CMT algorithm is available as an external library[5]. TLD is implemented in openCV[6]

[4] https://rabbitmq.com/
[5] https://github.com/gnebehay/CMT
[6] http://opencv.org/

and available in its Python wrapper. The correlation tracker is implemented in the dlib[7] library. Dlib is a toolkit for machine learning, however, it also offers image processing tools. The programming language independent feature of AMQP enables the communication between the Java-based WebSocket server, the broker, and the workers.

To forward tasks from the server to the workers both sides use the remote procedure call pattern [34]. The server calls the `remoteTrack` method with parameters specifying the IP address and port number of the RabbitMQ broker. Additionally, a worker-specific queue name, the payload that is holding information about the location of the object of interest in the first frame, the file name of the footage, and the length of the tracking process are provided. The server generates a correlation-id to identify incoming messages from the worker and publishes the id with the payload to the broker using the direct exchange method presented in the Section 4.4.1. The broker puts the task into the corresponding queue and creates a temporary callback queue using the correlation-id for publishing the results from the workers back to the server.

The tasks in the queue are load balanced between the workers registered to this queue using the fair dispatch method described in Section 4.4.2. The workers can be in two different states: idle and working. An idle worker is observing its registered queue for a new task to complete. The broker assigns each new task to one of the idle workers. This idle worker switches then into the working state. In the working state, the worker uses the provided payload information to open the corresponding video file and proceeds to the starting frame. At this frame, the bounding box of the object of interest is defined and passed to the tracking algorithm. In the next step, the worker iterates over the frames with the possibility of skipping n frames. Each time the tracking algorithm detects the location of the object of interest in a frame, the information is passed back to the server using the callback queue. The server forwards this information using the push ability of WebSocket back to the client. This technique enables nearly real time updates of the tracking state.

[7] http://dlib.net

The worker has completed the task when the last defined frame is tracked or the end of the video sequence is reached. In the following, the worker switches back into the idle state and is ready to perform a next task from the queue. The queue is getting longer when more clients are requesting a task to be performed. The more workers are running, the faster the tasks can be processed.

5.6 Summary of the Tracking Service

The tracking service consists of several components described in the previous sections. This section provides a brief summary of the function and coordination of the service and its components. Figure 5.2 shows the interaction of the different components. In order to use the service, the client first needs to connect to the WebSocket API. The client wrapper library can be used to simplify the connection and the communication process with the WebSocket API. When the connection is established the `uploadFile` method can be used to upload new video footage to the server. In the next step, the `track` method is used to track an object in an uploaded footage by providing the start time in milliseconds, an optional end time in milliseconds, the file name of the desired footage, the type of algorithm, the bounding box coordinates of the object of interest at the start position, and an optional frame skip parameter.

In the next step the server routes the tracking task to the RabbitMQ broker. The broker puts the task in the according queue depending on the provided tracking algorithm type. Worker applications process tasks from their queues and track the location of the object of interest in the following frames of the footage. The tasks get load balanced by the RabbitMQ broker if more than one worker of the same kind are active. The worker returns the result every frame via the callback queue of the broker back to the server. The server pushes the current result via WebSocket to the client. The worker finishes when the provided ending time-mark or the end of the footage has been reached.

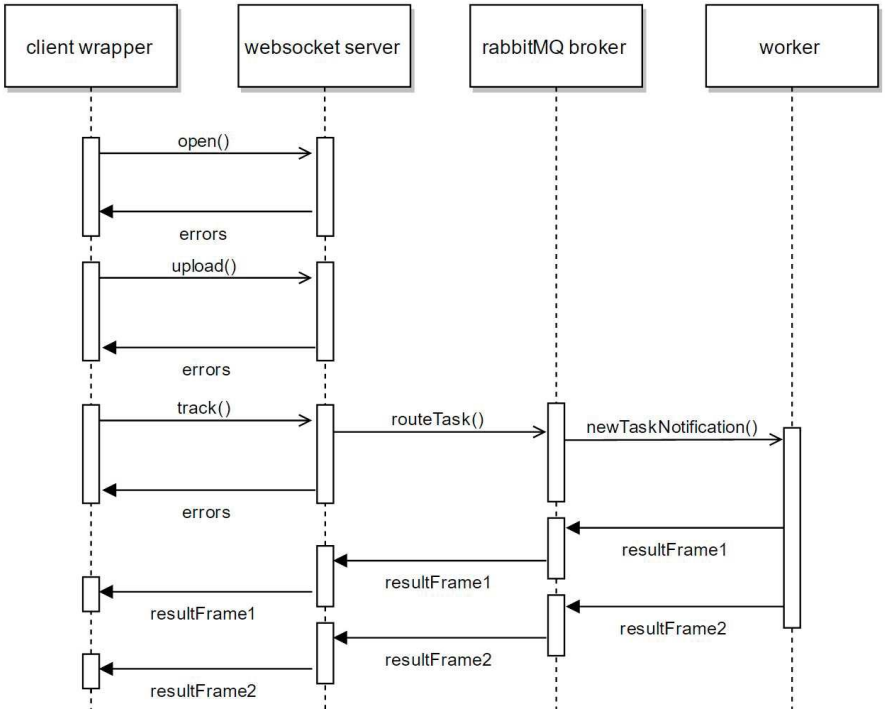


Figure 5.2: A sequence diagram showing the interaction of the different components. In this example only one non-specific worker is active.

5.7 Demo Implementation

The GlassFish server provides a demo web application to demonstrate the functionality of the tracking service. The application employs the client-wrapper in order to use the API of the WebSocket server. The user can select a videofile which is automatically uploaded. A file name gets the hash value of the file assigned[8] in order to determine if the file has already been uploaded to the server. If a file with this hash name already exists, the upload is skipped.

The user can skip through the video and drag a bounding box[9] around an object of interest (see Figure 5.3). When the selection is confirmed, the desired tracking algorithm is applied on the server. The current result is pushed from the server to the application on each processed frame and saved in an array.

[8]https://github.com/matthiasklan/Filehash
[9]https://github.com/matthiasklan/mediacropper

The application creates a formatting container that changes its dimensions and position over time to show the results (see Figures 5.45.55.6). This is done by assigning the position and dimension from the data array to the container. The API returns data in 0.1 second steps and CSS animations[10] are used to interpolate the transition between the steps when playing the footage.

Figure 5.3: Bounding box around an object of interest

5.8 Containerization of Components

Each component of the tracking service is containerized using Docker. One advantage is that most cloud providers support Docker for easy integration of the services for public use. Otherwise, Docker is the only prerequisite to be installed on a server or host machine. Docker runs the containers in an isolated sandbox mode. This guarantees that the service will run and behave the same on any machine, resulting in an easier development and testing process.

A `Dockerfile` defines instructions for configuring a specific container. Listing 5.2 shows an example `Dockerfile` for configuring a worker. The `RUN` instruction

[10] https://developer.mozilla.org/de/docs/Web/CSS/CSS_Transitions/Using_CSS_transitions

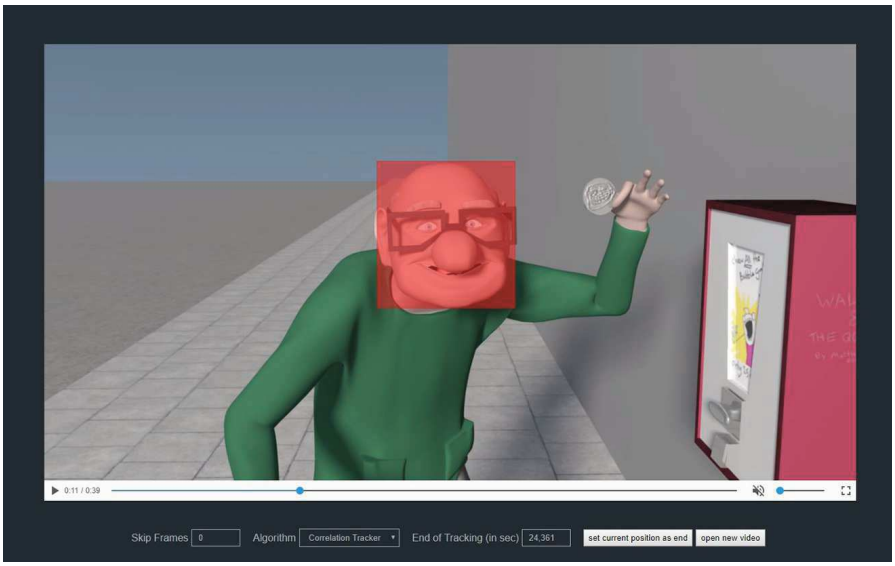


Figure 5.4: A red container visualizing the result at a certain frame

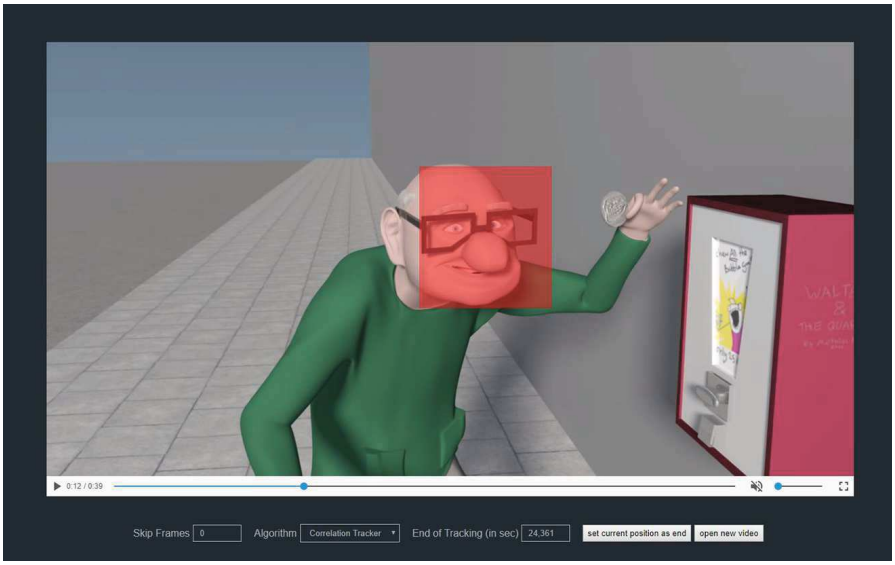


Figure 5.5: A red container visualizing the result at a certain frame

is used to install the necessary software on top of a minimal linux distribution in order to run a component of the tracking service. With the `COPY` instruction, relevant project files are copied into the container. `CMD` is the last instruction in a `Dockerfile` and specifies a custom command for starting a desired service in the container. The instructions need to be executed only once to build the container

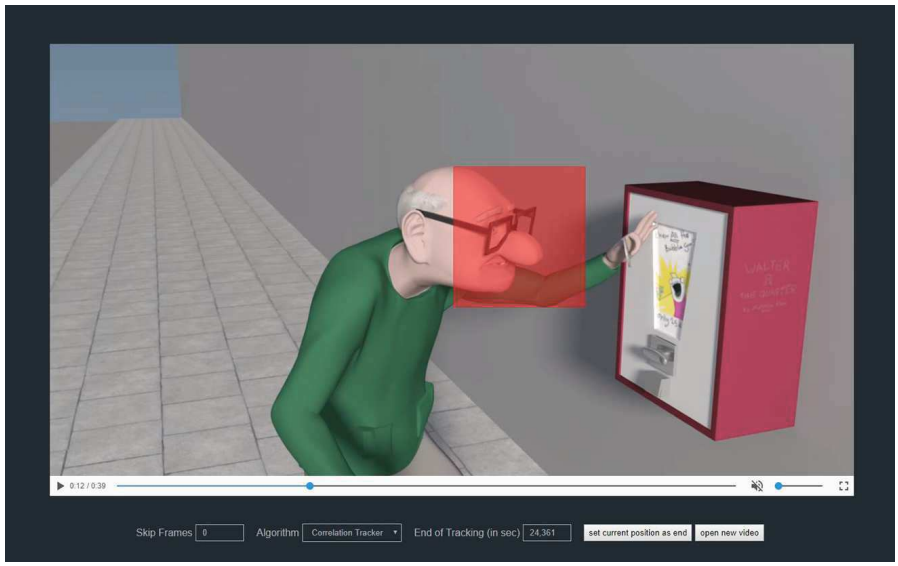


Figure 5.6: A red container visualizing the result at a certain frame

on a machine. The instruction steps are then cached. If a part of the instructions changes during development, only the altered part needs to be executed again.

Listing 5.2: Dockerfile for configuring a worker

```
1 # Use debian as the base distribution
2 FROM debian:sid
3
4 # Install dependency packages
5 RUN apt-get update
6 RUN apt-get install -y curl tar python python-dev python-pip
      python-virtualenv
7 RUN apt-get install -y python-opencv
8 RUN apt-get install -y python-matplotlib
9 RUN apt-get install -y build-essential cmake libboost-python-dev
10
11 # Install python library requirements
12 COPY requirements.txt requirements.txt
13 RUN pip install --no-cache-dir -r requirements.txt
14
15 # Copy compiled dlib library and app data
16 COPY ./dlib.so /usr/local/lib/python2.7/dist-packages/
17 COPY ./app /app
```

```
18
19 # Start the worker.
20 CMD ["python", "-u" , "/app/start.py"]
```

`Compose`[11] is a tool that allows to orchestrate multiple associated containers of an application. The `docker-compose.yml` configuration file defines all containers, their options, and dependencies. The RabbitMQ broker container exposes ports, so that worker containers and the WebSocket server container can connect to it. `Compose` also configures a shared disk volume to share the access to the uploaded video files between the containers.

5.9 Installation of the Tracking Service

In order to run the tracking service, Docker[12] has to be installed on the machine. To install and start the tracking service the `docker-compose up` command has to be executed within the project folder via the operating system's terminal. The API is then accessible via `wss://localhost/cloudtracking/actions`. The client wrapper library is accessible via `https://localhost/cloudtracking/socketClient.js` and can be imported as a JavaScript file into an own project. The demo application is accessible via `https://localhost/cloudtracking`.

To spin up a single container (e.g., by including a second worker for an algorithm), the command above has to be entered by the name of the container which is defined in the `docker-compose.yml` file (e.g., `docker-compose up cmt2`). To spin up more than two worker of the same type on the same machine, more additional containers, have to be defined in the `docker-compose.yml` file. This is done by duplicating a container definition and changing the name and IP to an unique value. To spin up additional container on a different machine, the IP of the application's machine has to be defined in the `vars.env` file located in the container's folder. Keep in mind that currently it is only possible to use workers

[11] https://docs.docker.com/compose/overview
[12] https://docker.com/products/docker

on other machines if the video files that should be tracked are already stored on the particular machine.

CHAPTER 6

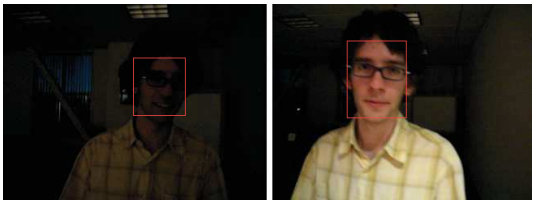
Experiments

This chapter deals with the experimental evaluation of the three implemented tracking algorithms in the web-based service. First, we conduct a performance evaluation in terms of accuracy and computation time of the algorithms. In the following sections, we perform experiments aiming at the reduction of computation time while still maintaining accuracy. These experiments investigate the skipping of several frames and the reduction of the footage's resolution.

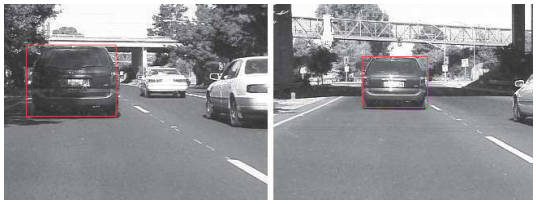
6.1 Performance evaluation

In this section the three tracking algorithms implemented in the service are evaluated with respect to their performance in terms of accuracy and computation time. A visual tracker benchmark website[1] offers sequences of footage with an annotated ground truth file. The data in the ground truth file represents the optimal bounding box coordinates of the object of interest in each frame. The sequences are a collection gathered from different tracking algorithm papers. Each of the sequences has different characteristics which represent challenging object tracking problems, such as illumination variations, scale variations, occlusions, deformation of objects, or motion blur (see Figure 6.1).

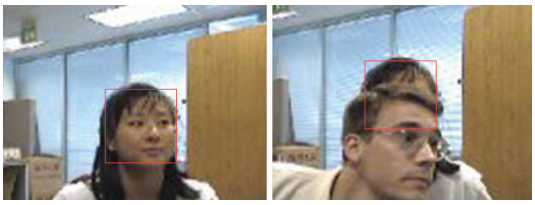
[1] http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html



(a) Illumination variance of the scene



(b) Scale variations of the object of interest



(c) Partial occlusion of the object of interest

(d) Deformation of the object of interest

(e) Motion blur in the scene

Figure 6.1: Different challenging characteristics of the evaluation sequences.

For this evaluation, each algorithm tracks a subset of 20 sequences (see Figure 6.2). They were selected because they have different characteristics [9] and duration. The shortest sequence has a duration of 71 frames. The longest a duration of 1,918 frames. The mean duration is 643 frames and the standard deviation is 503 frames. The smallest resolution is 128×96 pixels. The highest resolution is 768×480 pixels. The mean resolution is 486×344 pixels.

The results are compared with the ground truth data. To measure the trackers' accuracy, two different evaluation methods are used (see Figure 6.3). These methods are used by the prominent benchmark paper *Online object tracking: A benchmark* [47]. The first evaluation employs the Euclidean distance (see Equation 6.1) to measure the distance d between two points. In this case, the distance between the center of the ground truth bounding box α and the center of the computed

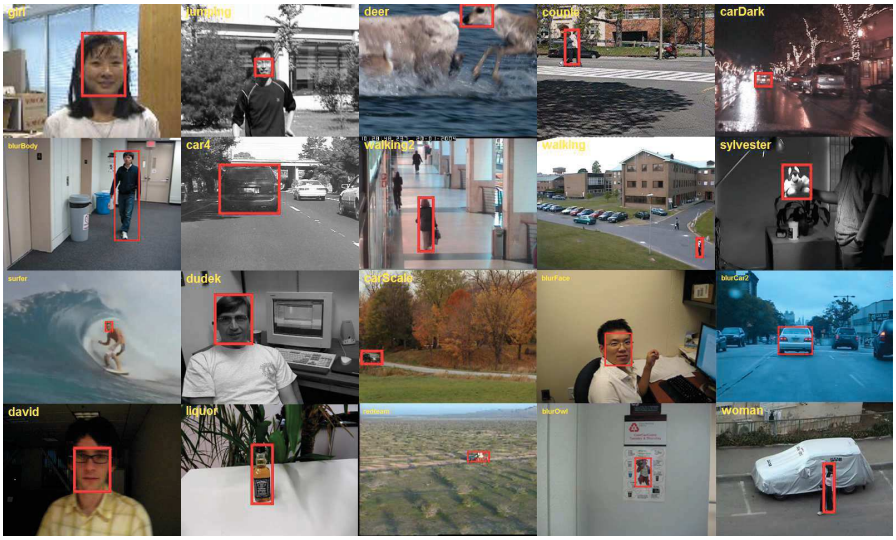


Figure 6.2: still frames of the 20 sequences used for evaluation.

bounding box t for each frame are taken into account. The larger the distance, the more inaccurate is the algorithm tracking of the object. The second evaluation measure calculates an overlap score (see Equation 6.2), a ratio S between the intersection $\bigcap$ and the union $\bigcup$ of the ground truth bounding box area r_α and the computed bounding box area r_t.

$$d(t, \alpha) = \sqrt{(t_x - \alpha_x)^2 + (t_y - \alpha_y)^2} \tag{6.1}$$

$$S = \frac{|r_t \bigcap r_\alpha|}{|r_t \bigcup r_\alpha|} \tag{6.2}$$

To visualize the performance of the algorithms a `precision` and `success plot` are generated, which are also introduced in the benchmark paper *Online object tracking: A benchmark* [47]. The `precision plot` (see Figure 6.4) shows the ratio of all frames that have a smaller distance than a given threshold, whereas the `success plot` (see Figure 6.5) illustrates the ratio of all frames that have an overlap smaller than a given threshold. The ratio of frames with bounding boxes within a maximal distance of 20 pixels to the ground truth (adopted from the benchmark [47]) is taken into account to rank the algorithms according to their

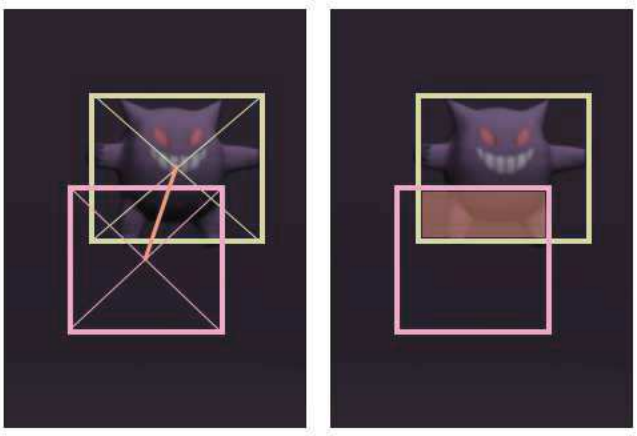


Figure 6.3: The Euclidean distance and overlap(orange) between ground truth(green) and the computed result(red).

precision. To measure the overall performance, the area under the curve (AOC) is used.

The success plot is more precise for ranking since it compares the percental overlap of the bounding box. That is why this method is independent of the footage's resolution. If the resolution is small, the Euclidean distance in the precision measurement should be considered stricter. Higher distances in a small resolution represent a higher inaccuracy. This is not considered by this evaluation. Therefore, in the later sections we only discuss the success plot.

The precision plot indicates the CMT as the most precise algorithm. 73% of the bounding box centers are within a 20 pixel distance of the ground truth center. The Correlation Tracker tracks 61% of the bounding boxes within this distance. TLD achieves 46% only. The success plot shows that CMT and Correlation Tracker perform similar, especially the amount of frames where the bounding box has an overlap of 60% and more.

Table 6.1 lists the mean computation speed of the algorithms. For the computation an `Intel(R) Core(TM) i5-5200U CPU @ 2.20Ghz` Dual Core CPU was used. The Frames per second (FPS) were measured directly when a worker finished a sequence. The remote FPS were measured at the client between the time when the client requested a tracking task of a sequence and the time when

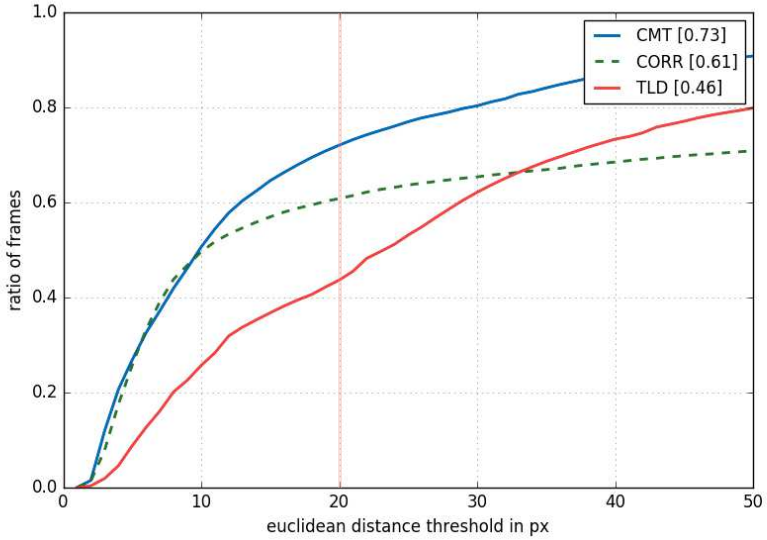


Figure 6.4: The precision plot of all 12,836 frames of the 20 sequences. The values in the brackets represent the accuracy at a threshold of 20px and rank the algorithms.

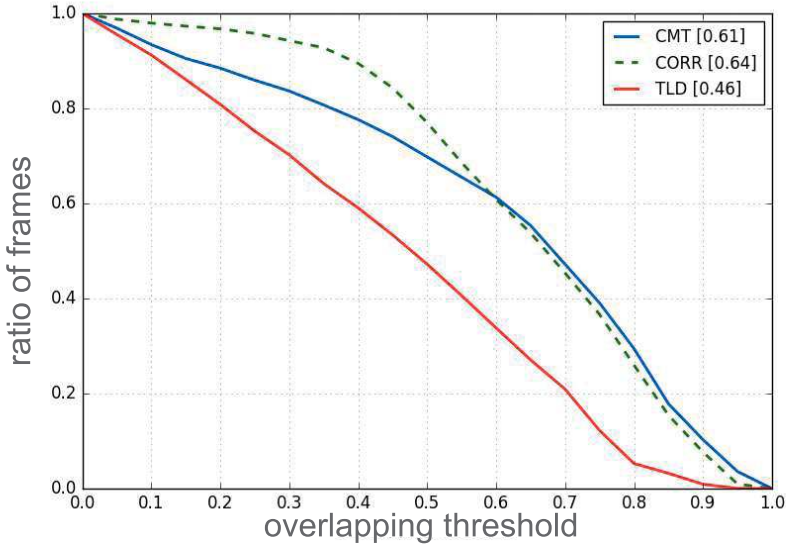


Figure 6.5: The success plot of all 12,836 frames of the 20 sequences. The values in the brackets represent the area under the corresponding curve and rank the algorithms.

the result for the last frame was received. The two different FPS measurements demonstrate the difference of the computation speed when tracking is performed directly on a machine and when tracking is requested remotely over the implemented service. The remote measurement was performed on a system where the service was running locally on the same machine as the client. This way a possible impact on computation time due to network latency is not considered in this measurement. However, the impact should not be significant due to the small data amount transferred over the permanent WebSocket connection.

The Correlation Tracker reaches 23 FPS measured at the worker. This is nearly a real time computation considering a video is commonly played back at 24 FPS. Using the service the Correlation Tracker still reaches 18 FPS. CMT is a little bit slower than the Correlation Tracker but still fast in comparison to the slow computation of TLD with 3/2.5 FPS.

Table 6.1 also lists the amount of frames with positively and negatively tracked objects. The Correlation Tracker tracks the objects of interest in every frame correctly (when the evaluation threshold is set to 20px). CMT does not track the objects in around 25% of the frames (FN), which is relative high in comparison to the other algorithms. True negative (TN) and false positive (FP) are always 0 since there is no frame without a tracked object in the ground truth. In other words, no sequence has frames where the object of interest completely disappears from the scene.

Table 6.1: The mean computation speed local/remote and the amount of objects of interest that are tracked falsely/correctly.

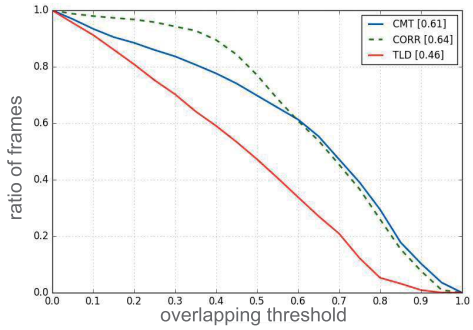
Algorithm	**FPS**	**remote FPS**	**TN**	**FN**	**FP**	**TP**
CMT	19.5	17	0	3790	0	9046
CORR	**23**	**18**	**0**	**0**	**0**	**12836**
TLD	3	2.5	0	225	0	12611

6.2 Experiment: Frame Skipping

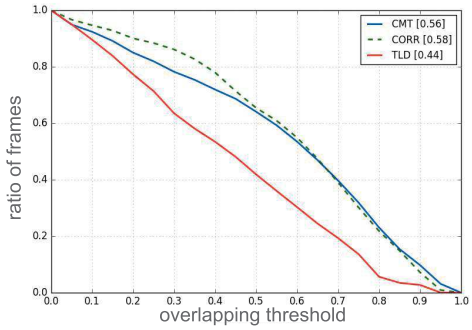
This experiment demonstrates how each implemented algorithm performs when 1, 4, or 12 frames are skipped between the frames of the 20 test sequences. Skipping frames may lead to a smaller overall computation time due to the smaller amount of frames that need to be tracked. In some cases the tracking does not need to be too frequent and the user can benefit from the faster computation. The downside is that trackers may get more inaccurate and lose track of the target due to more drastic changes in the scene. The gap between the tracked frames can be filled by interpolated values as shown in the demonstration application. However the tracking service does not return the interpolated results but the application developer has to implement the functionality by himself.

Figure 6.6 shows the results as success plots of the tracking when frames are skipped. According to the success score (see Table 6.2), the trackers do not loose too much accuracy. The success score of the Correlation Tracker drops by 25% when skipping 12 frames, whereas CMT's score drops by around 13% and TLD's by 22%. Table 6.2 also lists the average computation speed when skipping frames. Note, that the represented results also take frames that were skipped into account. When skipping one frame, the results are not twice as fast. The reason is the additional time it takes to perform the seek operation to the next position. However, the computation time gets faster, when more frames are skipped. According to the results, this experiment demonstrates that frame skipping is practicable to speed up the tracking process without too much loss of accuracy.

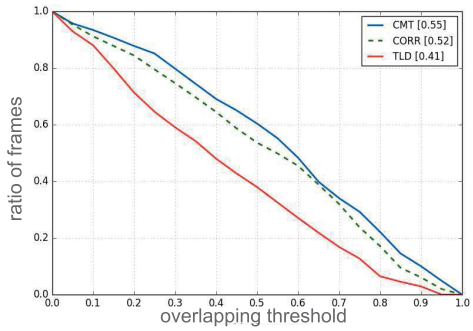
Table 6.2: The mean remote speed in *fps* and the success score (referenced as *ssc*) when 0, 1, 4, or 12 frames are skipped (referenced as *fs*) by the tracking algorithms. The bold values indicate the overall best performing algorithm.



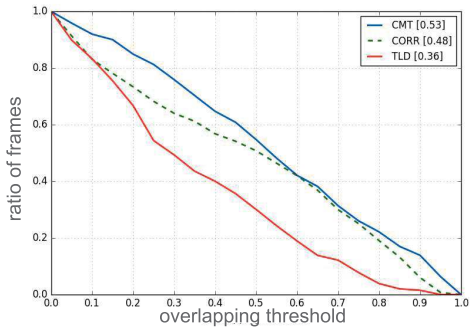
(a) no frame skipping



(b) skipping of 1 frame



(c) skipping of 4 frames



(d) skipping of 12 frames

Figure 6.6: The success plot without frame skipping (a) and the success plots with 1 (b), 4 (c), and 12 (d) frames skipped. The values in the brackets represent the area under the corresponding curve and rank the algorithms.

Alg.	**0 fs**	**1 fs**	**4 fs**	**12 fs**
CMT	17 fps, 0.61 ssc	19 fps, 0.56 ssc	66 fps, 0.55 ssc	103 fps, 0.53 ssc
CORR	**18 fps, 0.64 ssc**	**20 fps, 0.58 ssc**	**96 fps, 0.52 ssc**	**238 fps, 0.48 ssc**
TLD	2.5 fps, 0.46 ssc	3 fps, 0.44 ssc	15 fps, 0.41 ssc	70 fps, 0.36 ssc

6.3 Experiment: Reduction of Resolution

This experiment demonstrates how each tracking algorithm performs, when the resolution of the test sequences is reduced to 50% and 25%. All the sequences were scaled down in a preprocessing step, which is not included in the tracking service. On the one hand, a smaller resolution may lead to faster computation time because the trackers have less pixels to scan. On the other hand, the loss of details could lead to a lower precision in the identification of the accurate position of the target object.

Figure 6.7 shows the accuracy of the algorithms as success plots when the resolution is reduced. When reducing the resolution to 50% no significant loss in accuracy is detectable . Table 6.3 shows that no faster computation times were measured and the result get even worse. CMT drops from 17 remote fps to 15 fps and has a higher rate of frames with objects (3790 vs 4386) that were not detected. The Correlation Tracker drops from 18 fps to 14 fps while TLD has no significant drops. When the resolution is reduced to 25%, the accuracy drops notably according to the success plots in Figure 6.7 (CMT: 0.61 to 0.48, CORR: 0.64 to 0.44, TLD: 0.46 to 0.41). Overall the computation time is getting shorter (see Table 6.4). CMT reaches 24 fps remotely in comparison to the native 17 fps and the Correlation Tracker reaches 22 fps in comparison to the original 18 fps while TLD does not show any significant improvements. The number of objects that were not tracked is lower at 25% than at 50% of the resolution. This behavior could be caused by the object modeling of the tracking algorithms. High details in high resolutions lead to many features to identify the object, whereas too low details are resulting in identifying the objects with less features but still clearly separable from the background due to color changes and edges. The middle of those two extremes could lead to blurry weak features that are misinterpreted and are worse than no features at all.

The results show that only a drastic reduction to 25% of the original resolution leads to a slightly faster computation (only for CMT and the correlation tracker). A reduction to 50% leads to even worse performance in terms of computation time. In general, the faster computation is only relative, due to the computation time it

would take to reduce the resolution of the footage by the service.

Table 6.3: The mean computation speed local/remote and the amount of the objects of interest that are tracked falsely/correctly when the resolution is reduced to 50%.

Algorithm	**FPS**	**remote FPS**	**TN**	**FN**	**FP**	**TP**
CMT	20	15	0	4386	0	7952
CORR	18	14	0	0	0	12836
TLD	2.6	2.2	0	207	0	12629

Table 6.4: The mean computation speed local/remote and the amount of the objects of interest that are tracked falsely/correctly when the resolution is reduced to 25%.

Algorithm	**FPS**	**remote FPS**	**TN**	**FN**	**FP**	**TP**
CMT	29	24	0	3209	0	6046
CORR	26.5	22	0	0	0	12836
TLD	3	2.5	0	174	0	12662

6.4 Qualitative Analysis

This section deals with situations where the tracking algorithms have difficulties to keep track of the object of interest. These situations were observed when testing the algorithms with the implemented demo application.

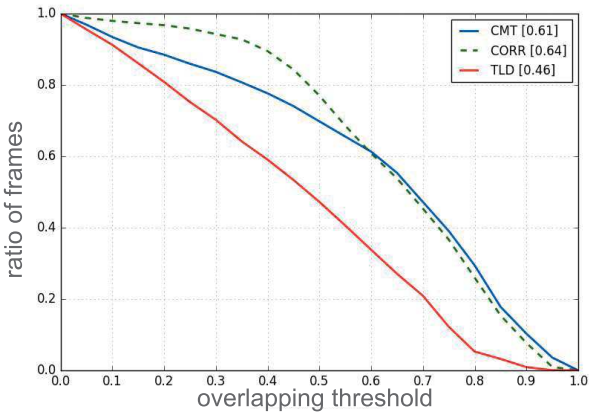
Fast movement of the object of interest leads to delayed updates of the bounding box, especially when using the Correlation Tracker. These delayed updates could be also caused by the lower frame-rate on which the bounding box is updated. On lower tempo this phenomenon is not observable. The CMT algorithm loses track and calculates an incorrect scale factor when fast movements occur, whereas TLD loses completely the track and locates the objects on wrong positions in the frame (see Figure 6.8).

CMT does not perform well, when the object of interest changes its size due to the change of distance to the camera. The algorithm again calculates the wrong scale

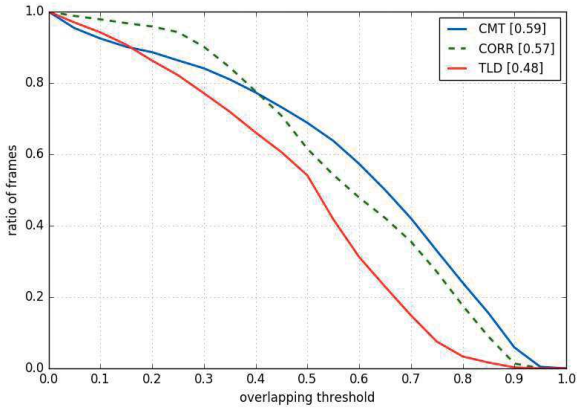
factor, whereas the Correlation Tracker is able to handle this type of situation well. TLD loses completely the track and the bounding box changes randomly its size even when no scale changes occur in the frame (see Figure 6.9).

All algorithms seem to have problems when objects of interest partially disappear from the scene (see Figure 6.10). The Correlation Tracker does not keep the correct scale factor, CMT often does not find the object in these type of situations, and TLD locates the object on wrong positions.

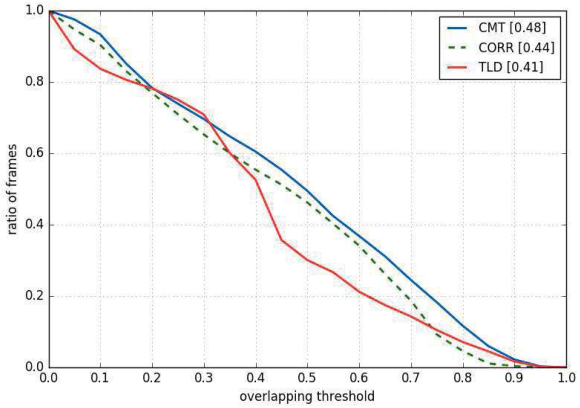
Each of the implemented algorithms has its typical characteristics when the object of interest is not located correctly. The bounding box of TLD jumps wildly in the scene from frame to frame and CMT calculates the scaling wrong or does not find the location at all. The Correlation Tracker is the only algorithm, where no remarkable failures of tracking are noted in comparison to the other trackers.



(a) original resolution



(b) 50% resolution

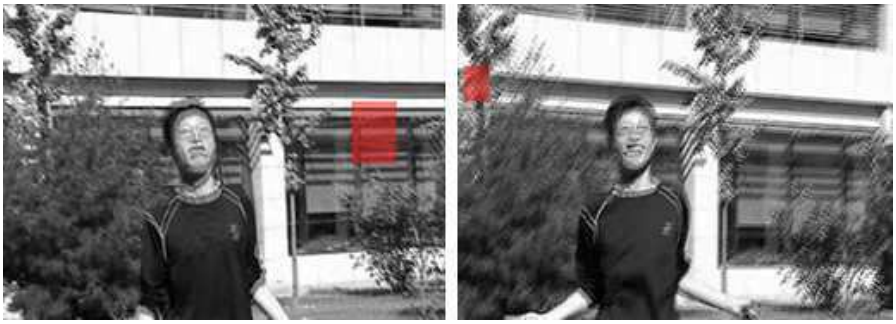


(c) 25% resolution

Figure 6.7: The success plot with original resolution (a), success plot with 50% resolution (b), success plot with 25% resolution (c)

(a) fast movement with CMT

(b) fast movement with Correlation Tracker

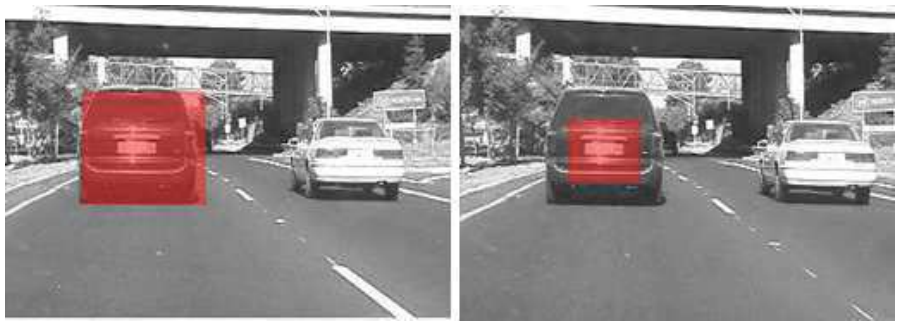


(c) fast movement with TLD

Figure 6.8: False tracking of the algorithms when the object of interest moves too quickly.

(a) object scaling with CMT

(b) object scaling with Correlation Tracker



(c) object scaling with TLD

Figure 6.9: The tracking behavior of the algorithms, when the object is scaling

(a) partial occlusion with CMT

(b) partial occlusion with Correlation Tracker

(c) partial occlusion with TLD

Figure 6.10: The tracking behavior of the algorithms, when the object is partially occluded

CHAPTER 7

Conclusion and Discussion

This chapter provides a summary of this thesis and compares the resulting implementation to other similar approaches which were introduces in the first chapter. Furthermore, the last section addresses open issues for further research. This covers, among other topics, the bad performance of the TLD tracker but also approaches for the optimization of the worker collaboration.

7.1 Summary

Software is moving from classic desktop applications to the web. Web-based applications allow for instant usage from any device via browser and without the need of installing additional software. Cloud computing is a software concept which separates specific application parts from the user and uses web technologies to connect the user with them. The user usually controls a graphical user interface which helps to communicate with the cloud parts. One advantage is the possibility to outsource heavy computation tasks to decoupled hardware in the cloud. Such tasks can be found in the computer vision domain, which tries to replicate the human vision in order to recognize specific features in a digital image. A specific computer vision task is object tracking, where algorithms try to locate an object of interest in consecutive frames of a video footage. This functionality can be used, for example, for traffic monitoring or surveillance.

Object tracking algorithms are mostly developed in programming languages not suitable for web application, such as *Matlab* or *C++*. The aim of this thesis was to design and implement a web-based cloud service for object tracking. Such a service would allow developers to easily create web-based object tracking applications without the need of putting effort into implementing tracking and infrastructure.

The implemented tracking service allows the user to upload a video footage to the server. Furthermore, the user can specify a location of an object of interest in any frame of the sequence. The service then performs the tracking for the following frames and returns the results back to the user. WebSocket is used for the communication between the user and the service. WebSocket's continual connection is preferable for time consuming requests such as object tracking. In this scenario the service is able to push new tracking results directly to the user, without the need to constantly ask if a new result is present. To easily use the exposed Application Programming Interface (API) of the service in new projects, a JavaScript library was developed. This library offers methods for connecting, uploading, and tracking. Additionally, the library allows to listen to certain events, such as receiving a new result or the upload status of a video file.

To enable the tracking functionality three tracking algorithms are currently implemented. These algorithms are using different techniques to retrieve the current position of an object of interest. TLD for example, is an algorithm which is using a learning database. This learning database is filled with information gathered from previous frames. This way the algorithm collects more and more knowledge about the object and can tune the performance for the following frames. The CMT algorithm in turn uses a novel way to eliminate falsely tracked points which do not belong to the object of interest by forming associated point clusters (see Section 4.2.2). The correlation tracker is the third implemented tracking algorithm. This algorithm converts frames into the frequency domain to efficiently correlate over them with a filter created from the object of interest.

These algorithms are not implemented into the service directly but as external worker applications. Using the Advanced Message Queuing Protocol (AMQP) the service is able to delegate the tracking requests to the workers. This implementation

design has several advantages such as the possibility to start multiple worker instances of the same type in order to load balance incoming tracking requests among each other. Another advantage is the programming language independence of AMQP. A good example of this advantage is an AMQP library named *RabbitMQ*, which provides implementations in nearly all common programming languages. This way it is possible to easily develop new worker applications which can implement new algorithms written in any language.

Finally, the different components of the tracking service are containerized using *Docker*. This has the advantage to ensure the correct execution on any machine due to encapsulated Linux distributions on which each container is running on. Additionally, the service can be easily setup by using automated installation tools named *Dockerfile* and *Docker-Compose*.

In the last chapter the implemented tracking algorithms are evaluated with respect to their performance in terms of accuracy and computation time. For the evaluation 20 annotated videos were tracked. The results were compared with the annotated ground truth data. Furthermore, two experiments were performed to examine the performance when multiple frames are skipped during tracking and when low resolution footage is used. The evaluation showed that the correlation tracker is the best performing algorithm in terms of accuracy and computation time. Finally, the results of the experiments demonstrated that the computation time can be speeded up while still having acceptable accuracy when skipping frames. However, this is not the case when shrinking the resolution.

7.2 Comparison with Related Work

The related work can be categorized into two groups. The first group are general and basic approaches and implementations of object tracking. The second group contains implementations which follow an approach similar to the one used in this thesis in terms of outsourcing the computation intensive tasks to an external hardware.

General object tracking approaches are algorithms written in different programming

languages, which can be integrated into own applications, also written in the same language. The main difference between the tracking web service of this thesis and such general approaches is that the implemented web service utilizes existing algorithms to create an enhanced tracking framework. The web service acts like a wrapper to offer advanced functionality in terms of cloud computing and easy integration into web applications. Developers are able to use this web service to build web-based applications and do not need to care about the communication and tracking aspects. The communication via WebSocket allows for a real time tracking experience directly in an internet browser. This way applications can be developed which do not need to be locally installed but are directly accessible over the internet.

There are some applications which follow similar approaches to outsource computer vision tasks into the cloud. Two existing services were already introduced in the first chapter. The first service is called *Wirewax*[1] and offers a web-based tool for creating interactive videos. The user can mark objects of interest in a video and make them clickable during the whole video. An application like *Wirewax* could be implemented using the web service of this thesis. The main difference is, that *Wirewax* is a more of a concrete application than a general tracking web service. The tracking data from *Wirewax* is bound to their system and can only be used with their interaction functionality. In contrast, the thesis's tracking web service lets the developer do anything with the tracking data. The data can be visualized, but also piped into another part of the software, for example, a database.

CloudCV[2] is the second related service, which offers a cloud-based computation of several computer vision tasks. This service is similar to the thesis's service in terms of the communication via WebSocket and the possibility for integration in web application. The main difference is the *CloudCV*'s missing object tracking functionality. Furthermore, WebSocket communication is only used between the service and the cloud and not between the client and the service. Moreover, the integration of *CloudCV* is only possible in the backend, hence, an extra backend application is required to develop a web-application utilizing *CloudCV*. In contrast,

[1] http://wirewax.com
[2] http://cloudcv.org

the thesis's tracking service can be integrated via JavaScript directly into a frontend applications and exchange data in real time using WebSocket. This communication would require for extra implementation steps when using *CloudCV*.

7.3 Discussion of Open Issues

One of the most obvious issues that should be resolved is the low performance of the TLD algorithm. The low performance was pointed out by the results of the evaluation in the previous chapter and does not confirm previously reported results by other benchmarks [15][47]. In these benchmarks the performance is much better in comparison to the evaluation results in this thesis. A future approach could be using a different implementation other than the one from *openCV* employed in this thesis. The *C++* implementation *openTLD* [3] could be promising, because the developer is the same one who also released the well performing CMT algorithm. The integration into the service is straightforward due to the programming language independent communication protocol of *RabbitMQ* between the service and the worker.

Another unresolved issue affects worker instances which are running on other machines than the WebSocket server. Distributed workers cannot remotely access the uploaded video files to perform tracking tasks. Currently, the files need to be copied manually to be able to perform hardware distributed load balancing. In this case, only already known and previously stored video files can be tracked by remote workers. *Docker* shares the upload folder of the WebSocket server with the worker containers. However, this works only if all containers have access to the storage disk. A possible solution could be to mount a global network directory where all relevant container would have access to. The files could be uploaded directly into the network directory and the worker instances could access the files remotely. Another way could be the usage of an ftp or file server for file exchange and distribution. The disadvantage for both variants is that they will expand the overall time-consumption of the tracking process. The reason is the additional download task from the worker instances. To overcome this sub-problem, the files

[3] https://github.com/gnebehay/OpenTLD

could be streamed from a remote file server. The streaming functionality would download only small chunks that are relevant for the current position. This way the system would not need to wait until the whole file is downloaded.

The next issue involves the scaling of the system. The current status of the service allows the manual only adding of additional worker instances. A better approach would be to automatically spin up extra workers whenever multiple concurrent tracking tasks are requested. This could be achieved by using a library like *docker-java*[4] to programmatically communicate with docker from the WebSocket server.

The last issue is more of an additional rather than missing functionality. Currently, the tracking requests are load balanced and thus assigned to free worker instances. An interesting approach would be to let multiple workers process a single tracking request. The idea is to split the tracking request in evenly parts according to the amount of frames. Each worker would process only a specific range of frames and the results would be than combined back together. On the one hand, this method would result in a shorter overall computation time. On the other hand, no remarkable increase in performance would be noticeable for the user in the first part due to the fact that the same worker processes the subsequent frames of a current processed frame. If two workers for example would track a video with this approach, the tracking could be finished after the half of the time. This would only apply if the second half of the video is processable as fast as or at least evenly fast like the first half. A further improvement could be a shared learning database for algorithms which are utilizing learning methods, such as TLD. This way the parallel workers could retrieve additional information about the object of interest, which is gathered by the other workers. This would lead to an extra boost of the performance.

[4]https://github.com/docker-java/docker-java

List of Figures

Bibliography

[1] Visual tracker benchmark. http://cvlab.hanyang.ac.kr/tracker_benchmark/benchmark.html. (Accessed on 11/26/2017).

[2] George B Arfken, Hans J Weber, and Frank E Harris. *Mathematical methods for physicists: a comprehensive guide*. Academic press, 2011.

[3] David S Bolme, J Ross Beveridge, Bruce A Draper, and Yui Man Lui. Visual object tracking using adaptive correlation filters. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 2544–2550. IEEE, 2010.

[4] David S Bolme, Bruce A Draper, and J Ross Beveridge. Average of synthetic exact filters. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2105–2112. IEEE, 2009.

[5] Sigismondo Boschi and Gabriele Santomaggio. *RabbitMQ Cookbook*. Packt Publishing Ltd, 2013.

[6] An overview of realtime libraries and frameworks. https://deepstream.io/blog/realtime-framework-overview/. (Accessed on 11/26/2017).

[7] Rita Cucchiara, Costantino Grana, Massimo Piccardi, and Andrea Prati. Detecting objects, shadows and ghosts in video streams by exploiting color and motion information. In *Image Analysis and Processing, 2001. Proceedings. 11th International Conference on*, pages 360–365. IEEE, 2001.

[8] Rita Cucchiara, Costantino Grana, Massimo Piccardi, Andrea Prati, and Stefano Sirotti. Improving shadow suppression in moving object detection

with hsv color information. In *Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE*, pages 334–339. IEEE, 2001.

[9] Visual tracker benchmark. http://cvlab.hanyang.ac.kr/tracker_benchmark/datasets.html. (Accessed on 11/26/2017).

[10] Understand the architecture. https://docs.docker.com/v1.9/engine/introduction/understanding-docker/. (Accessed on 11/26/2017).

[11] Overview of docker compose. https://docs.docker.com/compose/overview/. (Accessed on 11/26/2017).

[12] What is docker? https://www.docker.com/what-docker. (Accessed on 11/26/2017).

[13] Anand Singh Jalal and Vrijendra Singh. The state-of-the-art in visual object tracking. *Informatica*, 36(3), 2012.

[14] Z. Kalal, K. Mikolajczyk, and J. Matas. Forward-backward error: Automatic detection of tracking failures. In *Proc. 20th Int Pattern Recognition (ICPR) Conf*, pages 2756–2759, August 2010.

[15] Z. Kalal, K. Mikolajczyk, and J. Matas. Tracking-learning-detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(7):1409–1422, July 2012.

[16] John B. Kennedy. "when woman is boss - an interview with nikola tesla". http://www.tfcbooks.com/tesla/1926-01-30.htm, 1 1926. (Accessed on 01/29/2017).

[17] Leonard Kleinrock. Analysis of a time-shared processor. *Naval research logistics quarterly*, 11(1):59–73, 1964.

[18] Stefan Kolb and Guido Wirtz. Towards application portability in platform as a service. In *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*, pages 218–229. IEEE, 2014.

[19] BPL Lo and SA Velastin. Automatic congestion detection system for underground platforms. In *Intelligent Multimedia, Video and Speech Processing,*

2001. Proceedings of 2001 International Symposium on, pages 158–161. IEEE, 2001.

[20] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[21] Peter Lubbers and Frank Greco. Html5 web sockets: A quantum leap in scalability for the web. *SOA World Magazine*, 1(1), 2010.

[22] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. In *IJCAI*, volume 81, pages 674–679, 1981.

[23] Emilio Maggio and Andrea Cavallaro. *Video tracking: theory and practice*. John Wiley & Sons, 2011.

[24] Abhijit Mahalanobis, BVK Vijaya Kumar, Sewoong Song, SRF Sims, and JF Epperson. Unconstrained correlation filters. *Applied Optics*, 33(17):3751–3759, 1994.

[25] Peter Mell and Tim Grance. The nist definition of cloud computing. *none*, 2011.

[26] How microservices and containers are changing applications | opensource.com. https://opensource.com/business/14/12/containers-microservices-and-orchestrating-whole-symphony. (Accessed on 11/26/2017).

[27] Lyudmila Mihaylova, Paul Brasnett, Nishan Canagarajah, and David Bull. Object tracking by particle filtering techniques in video sequences. *Advances and Challenges in Multisensor Data and Information Processing*, 8:260–268, 2007.

[28] G. Nebehay and R. Pflugfelder. Consensus-based matching and tracking of keypoints for object tracking. In *Proc. IEEE Winter Conf. Applications of Computer Vision*, pages 862–869, March 2014.

[29] G. Nebehay and R. Pflugfelder. Clustering of static-adaptive correspondences for deformable object tracking. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, pages 2784–2791, June 2015.

[30] M. Ozuysal, P. Fua, and V. Lepetit. Fast keypoint recognition in ten lines of code. In *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, pages 1–8, June 2007.

[31] tracker_benchmark/trackers at master · jwlim/tracker_benchmark. https://github.com/jwlim/tracker_benchmark/tree/master/trackers. (Accessed on 11/26/2017).

[32] Top 6 open source python application servers. https://blog.idrsolutions.com/2015/05/top-6-open-source-python-application-servers/. (Accessed on 11/26/2017).

[33] Rabbitmq - rabbitmq tutorial - work queues. https://www.rabbitmq.com/tutorials/tutorial-two-java.html. (Accessed on 11/26/2017).

[34] Rabbitmq - rabbitmq tutorial - remote procedure call (rpc). https://www.rabbitmq.com/tutorials/tutorial-six-python.html. (Accessed on 11/26/2017).

[35] Harshitha. K. Raj. A survey on cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(7), 2014.

[36] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European conference on computer vision*, pages 430–443. Springer, 2006.

[37] Rotoscoping in ae. https://library.creativecow.net/articles/oconnell_pete/roto/video-tutorial. (Accessed on 11/26/2017).

[38] Main page. http://www.idiap.ch/~odobez/human-detection/. (Accessed on 11/26/2017).

[39] Soharab Hossain Shaikh, Khalid Saeed, and Nabendu Chaki. Moving object detection approaches, challenges and object tracking. In *Moving Object Detection Using Background Subtraction*, pages 5–14. Springer, 2014.

[40] Mike Shema. *Hacking web apps: detecting and preventing web application security problems*. Newnes, 2012.

[41] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR'94., 1994 IEEE Computer Society Conference on*, pages 593–600. IEEE, 1994.

[42] Nikola Tesla. Radio power will revolutionize the world. *Modern Mechanix and Inventions*, page 2, 1934.

[43] C Trieloff, C McHale, G Sim, H Piskiel, J O'Hara, J Brome, K van der Riet, M Atwell, M Lucina, P Hintjens, et al. Advanced message queuing protocol protocol specification. amq-spec. *AMQP. org*, 2006.

[44] James Turnbull. *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[45] Vanessa Wang, Frank Salim, and Peter Moskovits. *The definitive guide to HTML5 WebSocket*, volume 1. Springer, 2013.

[46] G. Welch and G. Bishop. An Introduction to the Kalman Filter: SIGGRAPH 2001 Course 8. In *Computer Graphics, Annual Conference on Computer Graphics & Interactive Techniques*, pages 12–17, 2001.

[47] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Online object tracking: A benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013.

[48] Rui Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, May 2005.

[49] Alper Yilmaz, Omar Javed, and Mubarak Shah. Object tracking: A survey. *Acm computing surveys (CSUR)*, 38(4):13, 2006.

[50] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.